Analysis of current languages for developing distributed systems

Alen Suljkanović*, Stevan Gostojić†, Igor Dejanović†, Sebastijan Kaplar†
Typhoon HIL, Novi Sad, Serbia*
Faculty of Technical Sciences/Chair of Informatics,Novi Sad, Serbia†
alen.suljkanovic@typhoon-hil.com*, {gostojic, igord, kaplar}@uns.ac.rs†

***Abstract*—Nowadays we have various languages, framework and tools for specification and implementation of distributed systems. Nevertheless, developing a robust distributed system still poses a challenging task. Starting from the definition that a distributed system is a system where networked components communicate via passing messages, we wanted to explore and analyse languages for specifying the architecture of such system (Architecture description languages - ADLs), as well as languages for specifying protocols for communication between system's components. Our main focus was to find out what are the main similarities and differences between different ADLs, and between languages for protocol specification.**

I. Introduction

Coulouris et al. in [1] define distributed system as one in which hardware or software components located at networked computers communicate and coordinate their actions only by passing messages. This definition covers an entire spectrum of systems in which networked computers can be used. These networked computers may be spatially separated by any distance. They may be in the same room or even on separate continents.

In order to design a distributed system, first we must understand what are the fundamental building blocks of such system. In [1], Coulouris et al., state that to understand the distributed systems, we must provide answers to the following questions:

- What are the *entities* that are communicating in the distributed system?
- How do they communicate?
- What (potentially changing) roles and responsibilities do they have in the overall architecture?
- How are they mapped on to the physical distributed infrastructure (what is their placement)?

From a system point of view, entities that are communicating can be seen as *processes*, *nodes* or *threads*. But, from the programming point of view those entities are seen as *objects*, *components*, *web services*, etc.

Entities, in a distributed system, can communicate *directly* or *indirectly*. The main characteristic of direct communication is that entities which communicate are aware of each other and must exist in the same time. Examples of direct communication are: *interprocess communication*, *remote invocation*, *request-response protocols*, etc. In contrast, numerous techniques have appeared where the communication is indirect, allowing a greater decoupling between entities. Entities do not need to be aware of each other (*space decoupling*), or event exist at the same time (*time decoupling*). Examples of such techniques are: *group communication*, *publish-subscribe*, *message queues*, *tuple spaces*, etc.

How entities in a distributed system communicate is described by a protocol. Holzmann in [2] provides a full protocol definition:

- it defines a precise *format* for valid messages (a syntax),
- it defines the *procedure rules* for the data exchange (a grammar),
- it defines a *vocabulary* of valid messages that can be exchanged, with their meaning (a semantics).

Protocol is often most easily understood as a finite state machine (FSM) [2]. The FSM is 5-tuple *(X, I, O, N, M)*, where

- X is a finite set of states,
- I is a finite set of inputs,
- O is a finite set of outputs,
- N is a state transition function (N: $I \times X \implies X$),
- M is an output function (M: $X \times I \implies O$).

N and M define a behaviour of FSM. For given input, the output function will generate the output, and the state transition function will generate the new state of FSM.

There are several other ways of representing protocols, such as: *Petri nets* and *FIFO nets*. Their best attribute is simplicity, but as the size of the model increases the readability quickly becomes an issue.

Languages analyzed in this paper can be divided in two groups:

- Architecture description languages, and
- Languages for protocol specification and validation

II. Analysis

Our main focus during the analysis was not to decide which language is the best, but to understand what are the main similarities and differences between the ADLs, and between the languages for protocol specification.

Analysis of ADLs and the analysis of protocol specification languages are performed independently, and are described in Section II-A and Section II-B respectively.

A. Architecture description languages

One of the most essential parts of software creation is architecture design. Architecture of software solution has a

Language	Application domain	Dynamic reconf.	Supports heterogeneous platforms	Debugging tools	Visualization tools
Conic	General	Yes	Yes	Yes	Yes
Durra	Embedded systems	Yes	Yes	Yes	No
Darwin	General	Yes	Yes	Yes	Yes
POLYLITH	Large Ada distributed systems	Yes	Partiall	Yes	No
Reality	Large Ada distributed systems	Yes	No	No	Yes
Rapide	General	Yes	No	No	No
DREMS ML	Real-time distributed embedded systems	Yes	Yes	Yes	Yes

TABLE I
COMPARISON OF ANALYSED ADLS

direct effect on software's extensibility, modularity, fault-tolerance, portability, scalability, and many other features of quality software solution.

As stated in [3], definition of Architecture Description Languages is still vague, and no formal definition of such languages is provided by the community. However, an Architecture Description Language - ADL, can be defined as a language that provides features for modeling a software system's conceptual architecture, distinguished from the system's implementation.

In order to better understand ADLs, we analysed following languages:

- **Conic**[4] - a language-based environment for building distributed systems. It provides a large set of tools for program configuration, compilation, debugging, etc. The Conic consists of *Conic Module Programming Language* - used for programming individual task modules, and *Conic Configuration Language* - used for configuration description and hierarchical structuring of distributed systems.
- **Durra**[5] - a language and the runtime support system for developing distributed applications.
- **Darwin**[6] - declarative binding language which can be used to define hierarchic compositions of interconnected components. Darwin is a successor of Conic, and it is based on π-calculus.
- **POLYLITH**[7] - an environment for rapid prototyping of distributed applications, with primitives that aid debugging and evaluation.
- **Reality**[8] - an executable design language for distributed Ada systems. Its main purpose is to enable rapid prototyping of large distributed systems.
- **Rapide**[9] - an object-oriented language for designing the architecture of not only distributed systems, but software systems in general.
- **DREMS ML**[10] - a "wide spectrum" architecture design language for distributed computing platforms, based on Model-Driven Development approach. DREMS ML is part of a complex architecture DREMS (Distributed Real-Time Managed Systems), which provides a large set of tools for modeling, analysis, debugging, testing and maintenance of distributed systems.

Comparison of analysed ADLs is shown in the Table I.

Although these languages are used in different domains, the principles behind these languages are quite similar. We noticed that their most common traits are:

- *Component based model* - distributed system consists of a collection of modular and reusable components, with clearly defined interfaces. Common characteristics of components can be abstracted into a *component types* which can be used used to create multiple *component instances*, and thus enhancing the reusability of the components.
- *Separation of structure from behaviour* - both structure and behaviour of systems are described by the separate languages. For example, in Conic, the structure of the system is described by the Conic Configuration Language, while behavior of its components is described by the Conic Module Programming Language. In Durra, the definition of components and communication channels is separate from the definition of configurations, etc. This separation is really beneficial when considering deployed systems and evolution as it allows us to focus on change at the component level rather than on some finer grain[6].
- *Dynamic reconfiguration* - mainly due to separation of structure from behaviour, it is possible to change the system in the runtime by adding new, or removing existing components. This characteristic is very important for creating robust, fault-tolerant systems.
- *Support for heterogeneous platforms* - Heterogeneity is still one of the biggest challenges when it comes to distributed systems. Most of the languages from above have support for different platforms, though in some cases (POLYLITH) that support is only partial.

Lead by the fact that these languages share many common traits, even though they are build for different purposes and different domains, we feel that a unifying language for developing a distributed systems could be of a great benefit. However, there are a few general UML-based solutions like Object Managment Group's SysML[1]SysML is a general-purpose graphical modeling language for specifying, analyzing, designing, and verifying complex systems that may include hardware, software, information, personnel, procedures, and facilities[11]. But, according to [12] domain-specific approach would offer greater benefits than general UML-based approach.

B. Protocol specification and validation

In order to understand what are the common characteristics of languages for protocol specification, we analysed following languages:

[1]Object Managment Group's SysML - http://www.omgsysml.org/what-is-sysml.htm

Language	Protocol layering	Protocol representation	Message format	Data types
LOTOS	Yes	Finite state machine	Built-in	Abstract data types
Estelle	Yes	Finite state machine	Built-in	Built-in
RTAG	Yes	Attribute grammar notation	User-defined	User-defined
Mace	Yes	Finite state machine	Built-in	Built-in
GAPAL	Yes	Finite state machine	User-defined	User-defined

TABLE II
COMPARISON OF ANALYSED LANGUAGE FOR PROTOCOL SPECIFICATION AND VALIDATION

- **LOTOS** (Language of Temporal Ordering Specification)[13] - a specification language that has been specifically developed for the formal description of the OSI (Open Systems Interconnection) architecture, although it is applicable to distributed, concurrent systems in general. Its main application is in OSI protocol and services specification.
- **Estelle**[14] - a Formal Description Technique, defined within ISO (International Organization for Standardization) for specification of distributed, concurrent information processing systems. Estelle is mainly used to describe protocols and services.
- **RTAG** (real-time asynchronous grammars)[15] - a language based on an attribute grammar notation. Its main purpose is to provide a software system for protocol development and experimentation, that can be used different environments.
- **Mace**[16] - a C++ language extension and source-to-source compiler that translates a concise but expressive distributed system specification into a C++ implementation. In this paper we focused on Mace's protocol specification mechanism.
- **GAPAL**[17] - a protocol specification language used by GAPA (Generic Application-Level Protocol Analyzer). Its main purpose is specification of messages that are being exchanged across the network.

As shown in the Table II, common characteristic of all of the languages mentioned above, is that they support protocol layering. This is really important, because in order to perform complex task, a protocol must perform a range of functions which would lead to complex implementations of the protocol. Because of this, protocols are divided into layers whereby each layer performs only a simpler sub-task of the complex task.

Another common characteristic of the analysed languages is that almost all of them use finite state machine to describe a protocol, except RTAG. RTAG uses BNF-like notation to describe a protocol.

The biggest differences between these languages lay in the way in which they define message formats and data types. For example, LOTOS uses *abstract data types* while GAPAL and RTAG allow user to define custom data types. *Abstract data type* does not indicate how data values are actually represented and manipulated in memory, but only defines the essential properties of data and operations that any correct implementation (*concrete data type*) is required to satisfy[13].

III. CONCLUSION

In this paper we analysed Architecture Description Languages, and languages for protocol specification in order to understand what are the main similarities and differences between the ADLs, and between the languages for protocol specification.

We found out that the ADLs share many common characteristics between themselves, just like the languages for protocol specification do.

Lead by this fact, we feel that a unifying language for developing a distributed systems could be of a great benefit. There are a few general UML-based solutions like SysML, but in our opinion domain-specific approach would offer greater benefits than general UML-based approach.

REFERENCES

[1] G. Coulouris, J. Dollimore, T. Kindberg, and G. Blair, *Distributed Systems: Concepts and Design*. Addison-Wesley Publishing Company, 5th ed., 2011.

[2] G. J. Holzmann, *Design and Validation of Computer Protocols*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1991.

[3] N. Medvidovic and R. N. Taylor, "A classification and comparison framework for software architecture description languages," *IEEE Transactions on software engineering*, vol. 26, no. 1, pp. 70–93, 2000.

[4] J. Magee, J. Kramer, and M. Sloman, "Constructing distributed systems in conic," *IEEE Transactions on Software Engineering*, vol. 15, no. 6, pp. 663–675, 1989.

[5] M. R. Barbacci, C. B. Weinstock, D. L. Doubleday, M. J. Gardner, and R. W. Lichota, "Durra: a structure description language for developing distributed applications," *Software Engineering Journal*, vol. 8, no. 2, pp. 83–94, 1993.

[6] I. Georgiadis, J. Magee, and J. Kramer, "Self-organising software architectures for distributed systems," in *Proceedings of the first workshop on Self-healing systems*, pp. 33–38, ACM, 2002.

[7] J. M. Purtilo and P. Jalote, "An environment for prototyping distributed applications," in *Distributed Computing Systems, 1989., 9th International Conference on*, pp. 588–594, IEEE, 1989.

[8] F. C. Belz and D. C. Luckham, "A new approach to prototyping ada-based hardware/software systems," in *Proceedings of the conference on TRI-ADA'90*, pp. 141–155, ACM, 1990.

[9] D. C. Luckham, J. J. Kenney, L. M. Augustin, J. Vera, D. Bryan, and W. Mann, "Specification and analysis of system architecture using rapide," *IEEE Transactions on Software Engineering*, vol. 21, no. 4, pp. 336–354, 1995.

[10] T. Levendovszky, A. Dubey, W. R. Otte, D. Balasubramanian, A. Coglio, S. Nyako, W. Emfinger, P. Kumar, A. Gokhale, and G. Karsai, "Distributed real-time managed systems: A model-driven distributed secure information architecture platform for managed embedded systems," *IEEE software*, vol. 31, no. 2, pp. 62–69, 2014.

[11] O. M. Group, "What is sysml?." Accessed: 2016-12-12.

[12] S. Kelly and J.-P. Tolvanen, *Domain-Specific Modeling: Enabling Full Code Generation*. Wiley-IEEE Computer Society Pr, 2008.

[13] T. Bolognesi and E. Brinksma, "Introduction to the iso specification language lotos," *Computer Networks and ISDN systems*, vol. 14, no. 1, pp. 25–59, 1987.

[14] S. Budkowski and P. Dembinski, "An introduction to estelle: a specification language for distributed systems," *Computer Networks and ISDN systems*, vol. 14, no. 1, pp. 3–23, 1987.

[15] D. P. Anderson, "Automated protocol implementation with rtag," *IEEE Transactions on Software Engineering*, vol. 14, no. 3, pp. 291–300, 1988.
[16] C. E. Killian, J. W. Anderson, R. Braud, R. Jhala, and A. M. Vahdat, "Mace: language support for building distributed systems," in *ACM SIGPLAN Notices*, vol. 42, pp. 179–188, ACM, 2007.
[17] N. Borisov, D. Brumley, H. J. Wang, J. Dunagan, P. Joshi, C. Guo, and I. Nanjing, "Generic application-level protocol analyzer and its language.," in *NDSS*, 2007.