Network and Communication Technologies; Vol. 5, No. 2; 2020
ISSN 1927-064X E-ISSN 1927-0658
Published by Canadian Center of Science and Education

# A Comparative Study of Formal Approaches for Web Service Oriented Architecture

Meftah Mohammed Charaf Eddine[1]

[1] University of Echahid Hamma Lakhdar, Eloued, Algeria

Correspondence: Meftah Mohammed Charaf Eddine, 39400, Bp 421, Guemar, Eloued, Algeria. Tel: 213-66-189-6380

Received: October 7, 2020 Accepted: November 5, 2020 Online Published: December 31, 2020

doi:10.5539/nct.v5n2p15 URL: https://doi.org/10.5539/nct.v5n2p15

**Abstract**

The security in web service oriented architecture (WSOA) development has become a critical need and goal. WSOA as service-oriented architecture (SOA) designs the software as services and uses the services as platforms. Web services orchestration describes how web services can interact with each other from an operational perspective. Many languages allow expression of executable processes to implementing web services orchestration. These languages are used to describe how the interactions between multiple services are coordinated to achieve a goal. However, the operational semantics of each of the structures of these languages is not formally defined and they have limitations regarding the reasoning and the verification of the web services compositions. Several studies and approaches have been proposed in this context, are devoted to the formalization of web services orchestrations and allow some verification of their behavior; these approaches are partial solutions to the problem of development of the safe composition. We explore the advantages and limitations of more than fourteen approaches and research work. We propose a model for comparison between works, studies, and approaches in this field. The proposed model adopts the concepts of the formalization and the automation of development processes.

**Keywords:** formal approach, web application, model, architecture, WSOA, orchestration

## I. Introduction

According to a study by Gartner Group. In 2020, the Software as a Service (SaaS) market was estimated at nearly $ 110,5 Billion (US) worldwide. The growth of the SaaS solutions market is estimated at nearly 17.5% (compared to 2019).

Table 1. SAAS market growth (Gartner, 2020)

| | 2007 | ... | 2018 | 2019 | 2020 | 2021 | 2022 |
|---|---|---|---|---|---|---|---|
| SAAS | 5 Billion (US) | | 80,0 Billion (US) | 94,8 Billion (US) | 110,5 Billion (US | 126,7 Billion (US) | 143,7 Billion (US) |

The term “Web 2.0” used by Dale Dougherty in 2003, released by Tim O'Reilly in 2004 and consolidated in October 2005 in a famous article published in September 2005: “What is the web 2.0” (Tim, 2005).

The term “Web 2+” refers to some of the World Wide Web technologies and uses that have followed the original form of the web (MyInfo, 2005).

According to Andi Gutmans, Web 2 is based on three pillars: a service-oriented architecture (SOA), rich web applications (RIA) and a social aspect.


Figure 1. The three pillars of Web 2+

The web service: is a computer program of the family of web technologies ,allowing the communication and the exchange of data between applications and heterogeneous systems in distributed environments. synchronously or asynchronously. without human intervention. (W3C, 2005). The term “Web service” describes a standardized way of integrating Web-based applications using the XML, SOAP, WSDL and UDDI.

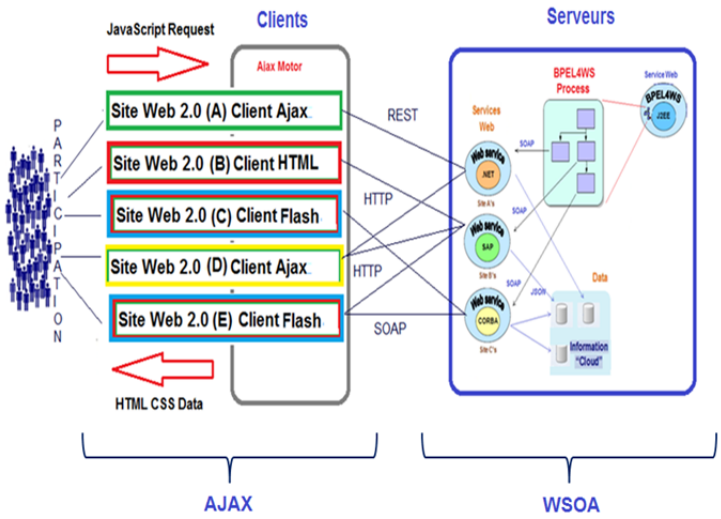


Figure 2. Web (2+) Application Architecture

A service, within service-oriented architectures, respects three properties:

(1) The service contract is exposed in an interface independent of any platform,

(2) The service can be dynamically located and invoked,

(3) The service is autonomous and knows how to maintain its current state.

*The choreography*

- Describes collaboration between services to accomplish a certain goal.

- Describes the different messages that pass between the different actors of a process (the services), the identity of which is not necessarily known.

- Enables point-to-point collaboration between multiple web services.

- It gives an abstract view of the exchanges within a process.

- It does not allow an execution, but it serves as a first specification for the concrete process (orchestration) to be performed.

- The choreography is decentralized coordination where each participant is responsible for a part of the workflow and knows his part in the flow of exchanged messages.

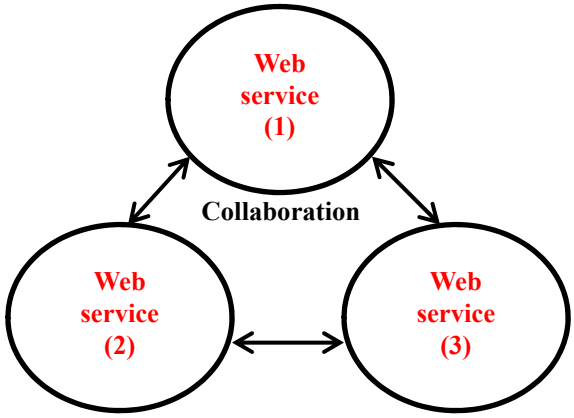


Figure 3. The choreography

Currently, the web services choreography is based on two standards, the WSCI and WS-CDL specifications.

*The Orchestration*

Orchestration describes how web services can interact together at the message level, including the order of interactions (messages) (often referred to as “business logic”).

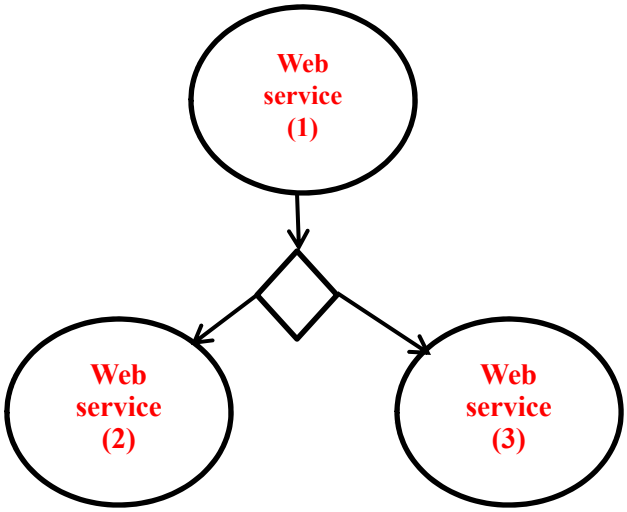


Figure 4. The Orchestration

The orchestration describes how web services can interact with each other from an operational perspective, with control structures.

-In the orchestration, a single process, called orchestrator, is responsible for the composition and controls the interactions between the different partner services that do not know this composition.

- Orchestration gives a concrete vision that allows the expression of an executable process.

This paper aims to accommodate a review of formal approaches and techniques that have been proposed in the Literature of WSOA (Web Service Oriented Architecture) . In addition, this paper will study and compare both the

strengths and downsides of all these approaches. Finally, we propose a model of comparison between the approaches and works of formalization and developments of web applications based on WSOA. Next sections will tackle these issues in detail.

## 2. Related Work

The development of Web 2+ applications and the use of web services-based architectures are subject to service consistency problems. Indeed, how can we be sure that applications, based on remote Web Services, and interconnected by networks which we know almost nothing about them, will result the correct composition of web services, to the correct application? For this, it is necessary to have a precise operational semantics of the behavioral description languages of these applications.

This operational semantics allows applying formal methods of a mathematical formalism defined precisely, syntactically and semantically. Depending on the method considered, the use of this formalism may be limited to the drafting of unambiguous specifications, or maybe extended to proof activities or even to programming itself; to verify certain behavioral properties of the studied system. However, the practical interest of a formal method is strongly dependent on the existence of tools implementing its formalism and possibly allowing automat some activities, such as the production of evidence.

To overcome the lack of formalism in the field of the web service oriented architecture (WSOA) development, several works are dedicated to the formalization of web services orchestrations and thus allow certain verifications of their behavior. This verification step will ensure a certain level of confidence in the internal behavior of an orchestration. In this section we will present and discuss these works:

### *2.1 LTSA-WS transition Systems*

The LTSA-WS (Labelled Transition System Analyser-WS) Transition System (Foster et al., 2003; Foster et al., 2005; Foster et al., 2006) is a systems verification approach. It verifies that the specification of a system satisfies the required properties of its behavior; through the comparison of two models, the specification model (design) and the implementation model.

A state transition system, or automaton in the broad sense, is an abstract machine model, used in theoretical computer science to simulate the progress of a calculation. It consists of the data of a set of states, and a set of transitions from one state to another. In other words, the formal definition of a state transition system is a couple:

$(S, \rightarrow)$ with $\rightarrow \subset S \times S$, where S is the set of states and→ is the transition relation.

If $(p, q) \in \rightarrow$ are two states, it means that there is a transition from p to q.

The system of transitions is deterministic if and only if → is a “function”, non-deterministic .

Transition systems play an important role in the recognition of formal languages, especially in their classification.

Common examples:

-Turing machine;

-Finite state machine;

-Petri net;

-Oriented graph;

A system in LTSA is modeled as a set of finite state machine interactions. The required properties of the system are also modeled as state machines. LTSA performs accessibility analysis of the composition of a comprehensive search on violations of desired properties. More formally, each component of a specification is described as a labeled transition system (LTS), which contains all the states a component can reach and all transitions.

On the other hand, the EFS (Extended Finite State) also allows the verification of properties of safety and very general liveliness (no blocking paths, any state is attainable).

- During the design phase, the approach uses UML (sequence diagrams) to describe how the different web services of orchestration are used and how they interact.

- The set of scenarios obtained are then composed and synthesized to generate a behavior model in FSP (Finite State Process), itself compiled into an LTS.

- During the implementation phase, the approach uses BPEL4WS to design the orchestration of the web services used.

*2.2 The Petri Nets*

Petri nets are the subject of work (Hamadi & Benatallah, 2003) aimed to formalizing web services orchestrations. This approach allows the expression of certain operators specific to the management of the control flow, such as the sequence, the alternative, iteration, parallelism, discrimination or selection. The Petri net describing an orchestration can then be analyzed and even compared with other Petri nets.

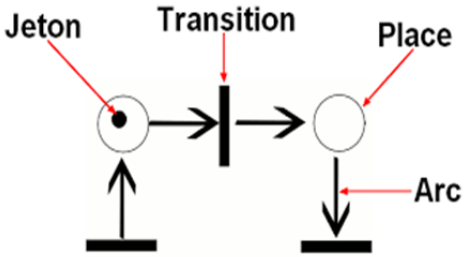


Figure 5. The components of the Petri nets

- A place is represented by a circle
- A transition by a line
- An arc connects a place to a transition

A Petri net is a way to:

- Modeling the behavior of dynamic systems with discrete events.
- Description of existing relationships between conditions and events.

*2.3 Process Algebras*

Process Algebra is a category of formalisms that describe and analyze competing or distributed systems (processes) behaviors. Different process algebras have been defined such as LOTOS (LOTOS 1989), π-calculus, the calculation of communicating systems (CCS). These process algebras are usually represented by a system of labeled transitions (LTS) where the transition relation is defined by a collection of rules and axioms. The main features of process algebras are:

- Specification and study of competing systems (communication, synchronization)
- Abstraction on behaviours,
- Synchronization mode (synchronous, RdV, complementary actions), etc.
- Composition mode (parallel, interlace, etc.),
- Semantic models (operational, traces, bisimulation, failure divergence).

*2.4 LOTOS/CADP*

LOTOS (BB88) is a formal specification language to describe communication protocols and distributed systems. It has been standardized by ISO/IEC in 1989 (ISO89). The design of LOTOS was motivated by the need for a language with a high abstraction level and strong mathematical bases that would allow complex systems to be described precisely and unambiguously, then analyzed using formal methods supported by appropriate software tools. LOTOS features two clearly separated parts:

The data part of LOTOS, intended to describe data structures, is based on the theory of abstract data types and algebraic specifications (especially the ActOne language defined by Ehrig and Mahr).

The control part of LOTOS is meant to describe the behaviour of concurrent processes that execute simultaneously, synchronize, and communicate using message-passing rendezvous. (CADP, 2020).

The CADP toolbox provides four main tools to handle LOTOS specifications:

- Caesar.adt is a compiler for the data part of LOTOS. It translates LOTOS to C by generating an implementation for all the sorts and operations defined in a LOTOS specification.
- Caesar is a compiler for the control part of LOTOS. It translates LOTOS to Petri nets, and then to C code (CADP, 2020).

LOTOS (Language of Temporal Ordering Specification), π-calculus, the calculation of communicating systems (CCS). These process algebras are usually represented by a system of labeled transitions (STE/LTS) where the transition relation is defined by a collection of rules and axioms.

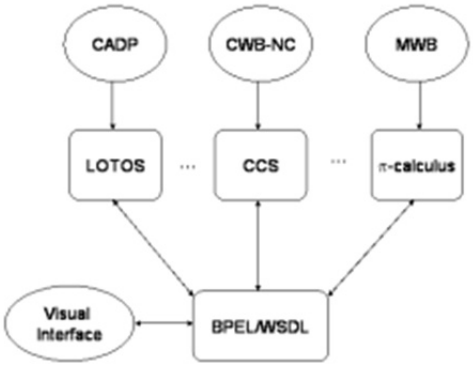


Figure 6.Web Services: a Process Algebra Approach (Andrea Ferrara)

The full syntax of a process is the following:

***process*** *P* (G0, ..., Gm) (X0:T0, ..., Xn:Tn) : *func := B*

***endproc***

Where B is the behavior of the process *P* and *func* corresponds to the functionality of the process

A tooled methodology for formal modeling and analysis of BPEL processes has been proposed by Rampacek (Rampacek et al., 2009). They defined a formalization of the BPEL semantics by algebra of time processes called (Atp).

Using the WSMod tool, the behaviors of a BPEL process are described by discrete-time transition systems. These models were analyzed by the Model-Checking technique using the CADP toolbox, in particular the Model-Checker Evaluator.

It is an approach (Salaun et al., 2004; Chirichiello & Salaun, 2005) to link, bidirectional, an orchestration described in BPEL4WS and its formalization in LOTOS (ISO / IEC, 1989). Equivalencies have been described so that behavior described in one language can be translated into the other and vice versa. On the other hand, thanks to the CADP tool, the approach makes it possible to reason about the formal description, expressed in LOTOS, and thus to verify the real orchestration, described in BPEL4WS. (Frédéric, 2007).

*2.5 π –calculus*

Like LOTOS, π-calculus (Milner, 1989; Milner, 1999) is a process algebra. These algebras are formal methods for modeling interactions between processes. For example, the following figure shows a process P that sends a value v to process Q through a transmission channel α. These algebras make it possible, by constructing a mathematical model containing certain descriptions of the model of analysis of the instantiation (those relating to the interactions), to guarantee the coherence of the model and the conformity of the program.

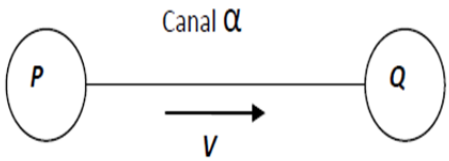


Figure 7. Inter process communication

*2.6 Temporal Theories*

These formalisms have been introduced to represent temporal knowledge, to specify domains of dynamic objects or to reason to solve problems. The work of (Rouached et al., 2006a; Rouached et al., 2006b) is based on one of these theories. Event Calculus (Kowalski & Sergot, 1986). They allow, initially, the formalization of a BPEL4WS orchestration thanks to a translation step, which makes it possible to represent an orchestration in the form of a set of predicates described in Event Calculus.

*2.7 Temporal Logics*

To describe the properties of proper operation of applications, temporal logics are well-adapted formalisms, in particular by their ability to express the scheduling of actions (events) over time. The properties descriptions in time logic present two important qualities (Manna & Pnueli, 1990):

- They are abstract, independent of the implementation details of the application,

- They are modular, ie the addition, the change or the deletion of a property does not call into question the validity of others.

There are usually two basic classes of properties on executions:

- Safety properties (under certain conditions, something “bad” will never happen),

- The properties of liveliness (under certain conditions, something “good” will eventually happen).

When choosing a temporal logic, several aspects must be considered, among which:

- Expressiveness (the ability of logic to express interesting classes of properties, such as safety or liveliness),

- Evaluation complexity (the complexity of algorithms to verify that a model satisfies a property),

- Ease of use (the ability to express properties concisely and naturally).

Optimization of one or other of these aspects can usually only be done to the detriment of others, the choice must be made through a judicious compromise (for example, if evaluation effectiveness is the most more importantly, then the expressiveness of logic will have to be limited). Besides, because of the diversity of existing temporal logic and the results present in the literature, it is not always easy to gather the relevant elements to choose a temporal logic adapted to a certain context.

*2.8 Abstract Service Design Language (ASDL)*

Abstract Service Design Language (Solanki et al., 2006) is a language based on Interval Temporal Logic (ITL) (Cau et al., 2006). Process algebra allows to reasoning over time constraints.

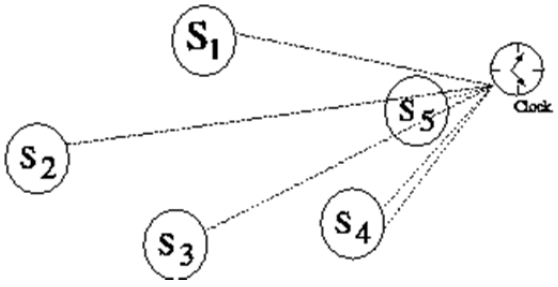


Figure 8. Broadcast of logical clock values

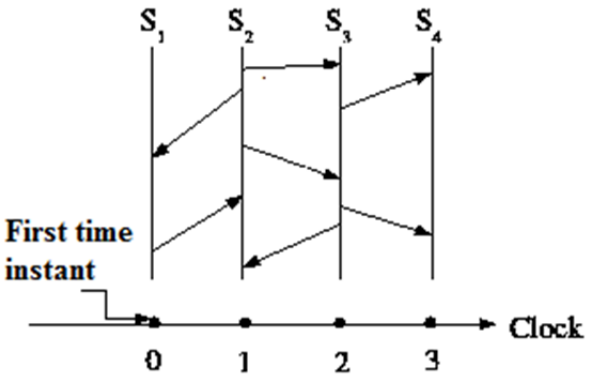


Figure 9. Clock values referenced by the services in the composition

ASDL aims at allowing the description of the behavior of a service, an orchestration and also the description of the protocols of interactions between the services. The objective of ASDL is to provide a notation for the design of service composition and interaction protocols at an abstract level. Therefore ASDL is a high-level formal language for reasoning and requires translation to a lower level orchestration language to be executed.

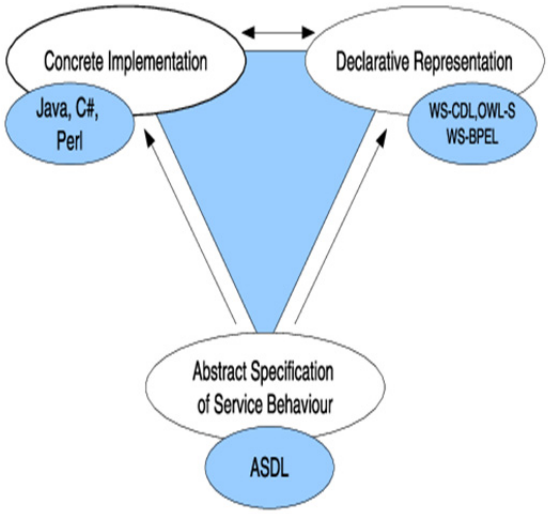


Figure 10. ASDL: A wide spectrum language for designing web services.

*2.9 FIACRE Language*

FIACRE (Farail & Gaufillet, 2008) is a formal language dedicated to modeling the behavioral and temporal aspects of systems, used for formal verification. It is a process algebra where hierarchical components communicate either through synchronous messages through ports or via shared variables. Ports and shared variables define the interface of a component. There are two types of components: leaf components (Process) and hierarchical components (Component).

Process: describes sequential behavior using symbolic transition systems (Henzinger & Manna, 1991). Thus, a process is defined by a set of states and transitions. Transitions may include non-deterministic actions (assignment, synchronization).

Component: describes the parallel composition of subcomponents. Also, a component can introduce variables and ports that will be shared by its subcomponents. At this level, real-time constraints and priorities can be associated with ports. A port p is associated with a time constraint in the form of a time interval I. This means that once the event p is activated, the execution must wait for at least the minimum bound of I and at most it's terminal. before disabling p or crossing the synchronized transition (Manna & Pnueli, 1990).

*Transformation: BPEL to FIACRE*

The static part of BPEL (WSDL) is modeled by global FIACRE types. The dynamic part of BPEL consists of the core business of the BPEL process and the handlers. It is modeled by a FIACRE root component that contains the composition of the FIACRE models associated with the primary activity and its handlers:

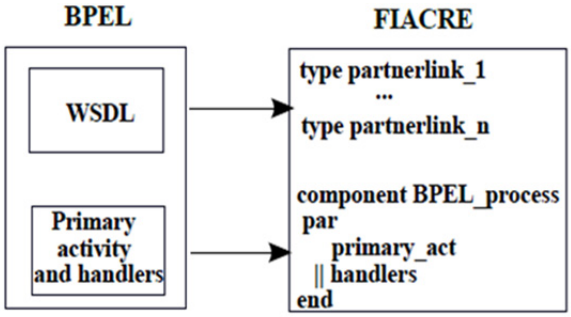


Figure 11. Structure of the transformation (BPEL-FIACRE)

Besides, the basic activities of BPEL are transformed into FIACRE processes while structural activities and handlers - able to contain other activities - are transformed into FIACRE components (Manna & Pnueli, 1990).

*2.10 Tree Logics Based on Actions*

The tree logic allows specifying properties on the execution trees resulting from the states or the actions of an STE (System of Transitions Labeled). They can be seen as extensions of modal logics with temporal operators expressing the potential or unavoidable accessibility of certain states and / or actions. The action-based tree temporal logics that are interpreted on STEs, which are the models associated with process-algebra languages (unlike the Kripke structures (Mateescu, 1998), based on states).

*2. 11 ACTL logic*

ACTL (Action Computation Tree Logic) (Nicola & Vaandrager, 1990) can be considered as the standard representative of tree logic for STE (action-based logic). ACTL logic contains three types of entities:

- Equity formulas (denoted α),

- Formulas on paths (noted $\Psi$),

- Formulas on states (noted $\Psi$).

These formulas respectively make it possible to characterize subsets of actions, paths, and states of an STE M = (S, A, T, s0). A path is a sequence of actions and states, connected by a transition relationship, forming a possible execution of a process. An ETS, therefore, corresponds to a set of paths.

The formulas are built using Boolean operators. The semantics of these operators is usual (set theory). The semantics of a formula α on an STE M = (S, A, T, s0) is defined by the interpretation ((α))

$\subseteq$ A, which denotes the subset of STE actions satisfactory α :

| | | | |
|---|---|---|---|
| $\alpha$ := a | [[a]] | = | a |
| \| false | [[false]] | = | $\emptyset$ |
| \| true | [[true]] | = | A |
| \| $\neg\alpha$ | $[[\neg\alpha_1]]$ | = | A $[[\alpha_1]]$ |
| \| $\alpha_1 \vee \alpha_2$ | $[[\alpha_1 \vee \alpha_2]]$ | = | $[[\alpha_1]] \cup [[\alpha_2]]$ |
| \| $\alpha_1 \wedge \alpha_2$ | $[[\alpha_1 \wedge \alpha_2]]$ | = | $[[\alpha_1]] \cap [[\alpha_2]]$ |

Path formulas are constructed using the next succession operator (denoted X) and the time operator until (denoted U). Given an STE M = (S, A, T, s0), the set of maximum execution sequences (that is to say the sequences ending in a state having no successor) is denoted as Path. The set of maximum execution sequences from an STE state can be described as:

$$\text{Path}(s) = s \xrightarrow{a_1} s_1 \xrightarrow{a_2} s_2 \ldots$$

avec $s \in S$

The semantics of formula on M is defined by the interpretation ((α)) Path, which denotes the subset of satisfying sequences $\Psi$ :

$\psi := X_\alpha \varphi$ $\quad [[X_\alpha \varphi]] = s_1 \xrightarrow{a_1} s_2 \ldots \mid a_1 \in [[\alpha]] \wedge s_2 \in [[\varphi]]$

$\mid \varphi 1_\alpha \, U \, \varphi 2$ $\quad [[\varphi 1_\alpha \, U \, \varphi 2]] = s_1 \xrightarrow{a_1} \ldots \xrightarrow{a_{i-1}} s_i \ldots \mid i \geq 1 \wedge s_i \in [[\varphi 2]] \wedge$
$\forall j \in [1, i-1]. a_j \in [[\alpha]] \wedge s_j \in [[\varphi 1]]$

$\mid \varphi 1_{\alpha 1} \, U_{\alpha 2} \, \varphi 2$ $\quad [[\varphi 1_{\alpha 1} \, U_{\alpha 2} \, \varphi 2]] = s_1 \xrightarrow{a_1} \ldots \xrightarrow{a_{i-1}} s_i \ldots \mid i \geq 2 \wedge s_i \in [[\varphi 2]] \wedge$
$a_{i-1} \in [[\alpha 2]] \wedge s_{i-1} \in [[\varphi 1]] \wedge \forall j \in [1, i-2]. a_j \in [[\alpha 1]] \wedge s_j \in [[\varphi 1]]$

*2.12 Diapason Approach*

The Diapason approach (Frédéric, 2007) proposes a way of describing web service-oriented architectures in a completely formal way, thus guaranteeing unity in the interpretation and execution of an orchestration. This formalization being kept from the description phase to the execution phase, it is then possible to guarantee that what is executed corresponds exactly to what is described, and whatever the virtual machine used. This formalization is carried out thanks to a language specific to the orchestration of web services: the π -Diapason language, which offers a power of expression superior to the current orchestration languages (for example BPEL4WS) and which is moreover extensible. The language π -Diapason is therefore defined in three distinct layers namely:

- A kernel layer, which corresponds to the π -calculus (base layer, allowing formalization and dynamic evolution),

- A workflow pattern layer, which allows enriching the operational semantics of the π -calculus (on-layer of the kernel layer, allowing the formalization of the concepts associated with the process management),

- A Web service-oriented layer, which specializes in the previous layer (overlay of the Workflow pattern layer, allowing the formalization of the concepts associated with the orchestration of web services).

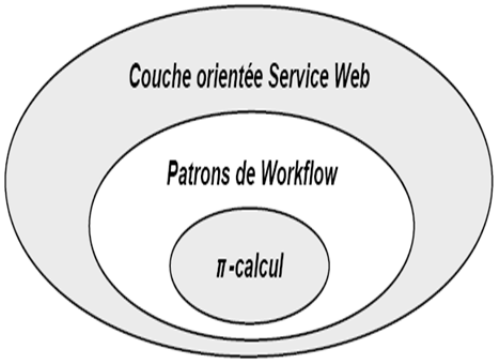


Figure 12. The language layers π -Diapason

In parallel with the definition of the π-Diapason language and to allow the verification of properties on an orchestration described with this one, there is also a language allowing to express in an intuitive way properties specific to a web service orchestration. This language is called Diapason * and extends the concepts of ACTL temporal logics (Nicola & Vaandrager, 1990) and ACTL * (Nicola & Vaandrager, 1990).

With these two languages, the Diapason architectural development process can be schematized as shown in the following figure; and it goes through different stages, some of them optional. The first step corresponds to the description of the architecture, thanks to the language π -Diapason and the description of properties related to this same architecture, thanks to the language Diapason *. This description can then be checked and modified as many

times as necessary until the desired architecture is obtained. Once the architecture is validated, it can be directly executed by a virtual machine interpreting the π -calculus. During execution, the π -Daposition description of the architecture can evolve to correct or prevent a potential error, to take into account new constraints or to take into account changes in the orchestration specifications. Before dynamically taking into account these changes, the new architecture can be checked again and modified as many times as necessary. Similarly, the expression of the properties relating to the running architecture can also change. Once the new architecture is validated, it is then necessary to check the current state of the running process to allow or not the taking into account of the new architecture while keeping its current state. This process can be repeated as many times as needed throughout the architecture lifecycle. These changes can be applied to a specific instance as on all instances of the running architecture. In the same way, they can be passed on or not on the global model of the architecture.

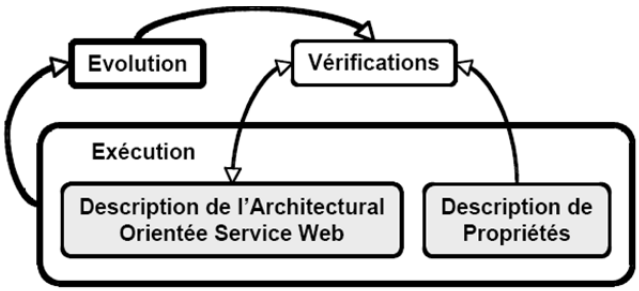


Figure 13. Web Service Architectural development process of the Diaposan approach

The different stages of the Diaposon approach are:

- The description of the architecture, thanks to the language π -Diapason,
- The description of properties related to this same architecture, thanks to the language Diapason *,
- Simulation and extraction of all possible execution paths of this architecture,
- Verification of different properties; this step can be followed by a modification of the architecture and be carried out as many times as necessary until obtaining the desired architecture,
- Execution of the validated architecture,
- The dynamic evolution of this architecture running; this step is accompanied by a new verification of the previous properties.

*2.13 CTT-B Approach*

This work (Meftah & Kazar, 2015) propose a formal approach for the development of safe web applications. This approach involves the generation of a web application on both sides (users' side (Ajax) and the web service side (Composition)) from formal specifications. In this approach the application is described in advance using graphical notations (CTT) (Fabio, 2012) and an automatic process is applied in order to translate them into formal specifications B (Abrial, 1996).

Grammar describing the syntax of CTT language:

T::= T >> T -- Enabling

| T()T -- Choice

| T II T -- Concurrent

| T |= | T -- Order independency

| (T) -- Optional process

| T (> T -- Disabling

| T |>T -- Interruption

| T (> T -- Disabling infinite process

| TN -- Finite process iteration
| Tat -- Atomic process

MODEL nameM
REFINES nameR
SETS . . .
PROPERTIES . . .
VARIABLES . . .
INVARIANT . . .
ASSERTIONS . . .
INITIALISATION . . .
EVENTS . . .
END

- EVENTS : The events are described using generalized substitutions (based on the weaker condition of Dijkstra)
(S)P : weakest precondition such that P is executed after the execution of S.
Substitutions in B models.

| | |
|---|---|
| (SKIP)P | P |
| (S1 II S2 )P | (S1)P ^ (S2)P |
| (ANY v WHERE E THEN S END)P | ¥ v.(E => (S)P) |
| (SELECT E THEN S END)P | E => (S)P |
| (BEGIN S END)P | (S)P |
| (x := E)P | P(x/E) |

Using the B refinement process, a set of rules refinement, operating on data and operations, is applied to the specifications obtained. These phases refinement are intended to make the final specifications close to the target implementation language chosen by so that the last coding phase becomes intuitive and natural. In general, the process of refinement is a manual task, relatively expensive, particularly in phase evidence. With character of refinement of these rules, an assisted refinement tool can be achieved, enabling reduction the cost of the refining process.

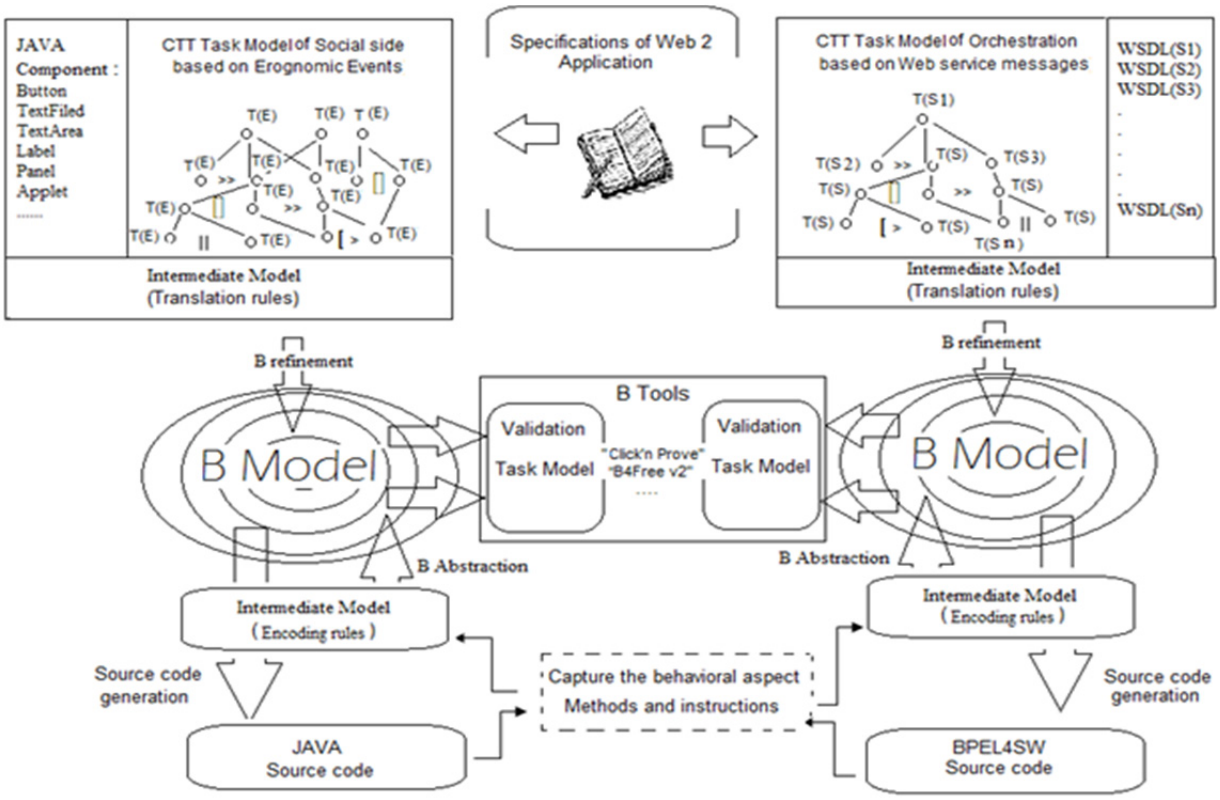


Figure 14. CTT-B method (Meftah and Kazar, 2015) .

*2.14 Model-Driven Approach (MDA-UML-S)*

There are Model-Driven Approaches (MDAs) are proposed to specify, validate and implement web services composition. A profile (named UML-S) has been defined to adapt UML to the service composition domain UML-S enables a clear and compact specification of the service composition, using class and activity diagrams. Class diagrams are used to describe the static part of the composition, ie the web services user interface, and activity diagrams are used to describe the dynamic part of the composition.

In this work (Nicola & Vaandrager, 1990), a model-driven approach (MDA) faithful to the principles defined by the OMG is proposed to specify, validate and implement the composition of web services. To achieve this goal while retaining the standard UML metamodel, a UML2 profile (or customization) of UML2 has been defined to adapt UML to the service composition domain. UML-S enables a clear and compact specification of the service composition, using class and activity diagrams. Class diagrams are used to describe the static part of the composition, ie the web services user interface. Activity diagrams are used to describe the dynamic part of the composition. The development process of this approach is presented in the following figure:

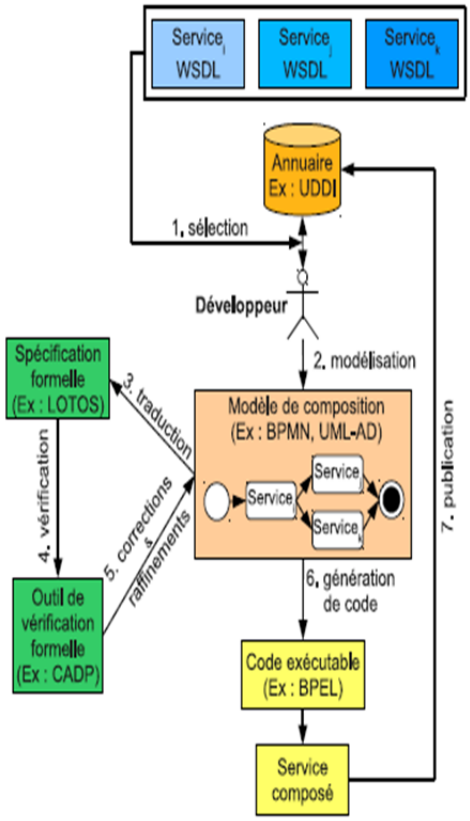


Figure 15. Model-Driven Approach (MDA-UML-S)

*2.15 Other Formalisms*

A Structured Activity Compensation (StAC) language (Butler & Ferreira, 2005) is defined to use it to specify the orchestration of activities for long-term transactions. This type of transaction uses compensation for exception handling. STAC considers sequential or parallel behaviors as well as compensation and exception handling. The notation B has been combined with the stAC language to specify the transaction data. stAC has been used to describe the semantics of a part of BPEL. On their side, Yuan et al. (Yuan & Zhong, 2006) has defined a new model called BPEL Flow Graph (BFG), which is an extension of the Control Flow Graph (CFG), to represent BPEL processes as a graphical model. This BFG model was used in the generation of test cases for BPEL. These test cases are built by combining the test paths generated by traversing the BFG model, and the data generated by constraint resolution. This BFG model makes it possible to specify the control flow of BPEL, the data flow but only part of the BPEL semantics.

In the literature, there are other works of modeling the composition of services by UML diagrams (OMG, 2008). We give, for example, the works of Cambronero et al. (Emilia & Gregorio, 2007) who used UML diagrams to describe BPEL process behaviors and associated temporal constraints. They also proposed an automatic method of transforming these UML diagrams into BPEL (LALLALI, 2009).

## 3. Comparative Study

Web services quickly became actors in complex processes. These processes can be approached in two ways: choreography and orchestration. Only the orchestration of web services gives a concrete vision that allows the expression of an executable process. The orchestrations are, for the majority, expressed thanks to the language BPEL4WS.

There is work based on STEs such as LTSA-WS (Foster et al., 2003; Foster et al., 2005; Foster et al., 2006) or process algebras, π-calculus (Milner, 1989; Milner, 1999) and π-Diapason (Frédéric, 2007) are, for their part, a mathematical formalism for the description and study of competing systems. They allow model representation accurately, using the operators of algebra to define each of the structures of a process. Among the work to formalize and analyze BPEL4WS, some as ASDL (Solanki et al., 2006) are based on the Internal Temporal Logic

(ITL) algebra (Cau et al., 2006) while others (Salaun et al., 2004; Chirichiello & Salaun, 2005) used LOTOS (ISO / IEC, 1989).

Temporal theories have emerged following the application of logic in the field of Artificial Intelligence. These formalisms were introduced to represent temporal knowledge, to specify domains of dynamic objects or to reason to solve problems; as. The works of (Rouached et al., 2006a; Rouached et al., 2006b) are based on one of these theories: Event Calculus (Kowalski & Sergot, 1986).

Temporal theories (The works of (Rouached et al., 2006a; Rouached et al., 2006b), FIACRE (Farail & Gaufillet 2008)), temporal logics, temporal logics based on actions and ACTL logic (Nicola & Vaandrager, 1990) appeared following the application of logic in the field of Artificial Intelligence. These formalisms were introduced to represent temporal knowledge, to specify domains of dynamic objects.

Other work proposed semi-formal approaches to the composition of web services such as Petri nets (Hamadi & Benatallah, 2003); and model-driven as (MDA-UML-S) (Nicola & Vaandrager, 1990).

After presenting the different works and languages related to web services orchestration, we found that several weak points were not addressed, or at least not satisfactorily.

Table 2. Comparative study

| *Related approaches* | *Strong points* | *Weak points* |
|---|---|---|
| LTSA-WS transitions systems | - The approach uses accessible standards (sequence diagrams, BPEL4WS) that mask the complexity of FSP.<br>- It provides a visual means (via the LTSA tool) to compare the design model and the implementation model while checking for no violation of general properties. | - The approach disengages the design and implementation phases to regroup them, once they are independently completed. In case of non-coherence (concordance) of traces, the implementation is totally to resume: the verification phase is then very / too late.<br>- UML being semi-formal and BPEL4WS having no formal semantics, it is impossible to prove that the generated FSP corresponds exactly to what is described. In this sense, comparisons and verifications can only be partial.<br>- The BPEL4WS specification can be interpreted by different execution engines, nothing to prove that they have the same semantics of interpretation. In fact, what is implemented corresponds (partially) to the needs defined during the design, but not necessarily to the reality of the execution |
| Petri nets | - This approach formalizes some operators used for the orchestration of web services in the form of a Petri net and thus allows certain verifications. | - This approach does not offer a means of expression close to the standards of the domain of Web services.<br>- Secondly, the formalized behavioral structures are very limited and do not allow the description of complex orchestrations.<br>- Petri nets offer simulation mechanisms to analyze processes but do not allow any execution of these. The approach must, therefore, be completed by a translation phase to an executable language, for example, BPEL4WS. Since the latter is not formal, no guarantee can be given as to the exact correspondence between the operators supported by the approach and their translation into BPEL4WS. |
| ASDL process algebras | - ASDL has a clearly defined semantics since it is based on process algebra. It allows the reasoning of a process.<br>- His high degree of abstraction allows him to describe an orchestration at different levels | - ASDL has a complex syntax, far from the standards of the Web services domain.<br>- Interfacing at a high level of abstraction requires another (lower level) language to perform an orchestration described with ASDL, such as BPEL4WS. This |

| | | |
|---|---|---|
| | (services, orchestration, protocols). | translation, free of all formalism, does not ensure that what will be executed corresponds exactly to what ASDL allowed to reason before. Moreover, it is not certain that everything that has been formalized finds an implementation structure (moving from a high to a lower level of abstraction). |
| LOTOS / CADP | - The strong point of this approach is the power of expression of the properties thanks to the CADP tool. Indeed, this tool allows the expression of properties of safety and vivacity specific to a given process. | - Despite its power of expression, CADP remains a complex tool to master. Indeed, it is not trivial to define a property and even less to express it, especially since the syntax of the language does not refer to the concepts of orchestration of Web services.<br>- This type of formalization does not ensure consistency between what is verified and what is executed.<br>- The approach imposes, a translation of process algebra to a language without formal semantics so that orchestration is executed. This recurrent step introduces a loss of semantics, amplified by the ambiguities that can be added by the different implementations of the BPEL4WS engines. |
| π -calcul | - The peculiarity that characterizes π-calculus is the introduction of the concept of mobility, that is to say, the possibility of dynamically modifying the links between different processes as well as moving behavior from one process to another. | - π-calculus presents a complex syntax, far from the standards of the Web services domain.<br>-Second, the fact of interfacing with a high level of abstraction requires another language (lower level) to execute an orchestration described with π-calculus. This translation, free of any formalism, does not make it possible to ensure that what will be executed corresponds exactly to what π -calculus allowed to reason before.<br>-Moreover, it is not certain that everything that has been formalized finds an implementation structure (moving from a high to a lower level of abstraction). |
| Temporal theories | - The approach allows the verification of functional and non-functional properties.<br>- It allows the verification of a BPEL4WS orchestration at a static level (before execution) and throughout the execution of an orchestration. | - The description of the properties in Event Calculus requires some expertise, far removed from the standards of the domain of Web services.<br>- The translation phase between BPEL4WS and its formalization language remains as in the other approaches, which induces the same restrictions, ie potential loss of semantics. These are partially offset by the verification phase throughout the execution. However, the detection of drifts during the execution occurs once the latter has actually arrived. |
| Temporal Logic and ACTL Logic | - Their ability to express the scheduling of actions (events) over time. | - The optimization of one or the other of these aspects (the expressivity, the complexity of evaluation, the ease of use) can generally be done only to the detriment of others.<br>- Due to the diversity of the existing temporal logic and the results present in the literature, it is not always easy to gather the relevant elements to choose a temporal logic adapted to a certain context. |
| Diapason approach | - Describe web service-oriented architectures in a completely formal way.<br>- Ensuring unity in the interpretation and execution of an orchestration. This formalization being kept from the description phase to the execution phase, it is then possible | - The complexity of syntax and use of language.<br>-Some steps of the Diapason approach remain manual.<br>- The π-Diapason language does not take into account existential and universal quantifiers on actions.<br>- Do not take into account the so-called non-functional |

| | | |
|---|---|---|
| | to guarantee that what is executed corresponds exactly to what is described.<br>- The π-Diapason language, which offers a power of expression superior to the current orchestration languages (for example BPEL4WS) and which is moreover extensible. | properties. |
| CTT-B approach | - This approach involves the generation of a web application on both sides (users' side (Ajax) and the web service side (Composition)) from formal specifications.<br>- with character generic of these refinement rules, an assisted refinement tool can be achieved.<br>- Ascending and descending approach. | - Complexity and difficulty to deal with refinement concepts. |

## 4. Model of Comparison

In this section, we propose a model of comparison between the approaches and works of formalization and developments of web applications based on WSOA. The proposed model adopts six (06) aspects of comparison:

- ✓ Formalization of Architectural-side (server),
- ✓ Formalization of Technology-side (interface of clients),
- ✓ Development orientation (Ascending or Descending),
- ✓ Degree of formalization.
- ✓ Automation of development processes;
- ✓ Automatic code (BPEL4SW, JAVA) generation.

Table 3. The model of comparison

| *Approaches* | *Architectural aspect* | *Technological aspect* | *Ascending* | *Descending* | *Degree of Formalization* | *Automation* | *BPEL4SW Code* | *JAVA Code* |
|---|---|---|---|---|---|---|---|---|
| LTSA-WS transitions system | √ | × | × | √ | √ | × | √ | × |
| Pétri nets | √ | × | × | √ | × (semi formal) | × | √ | × |
| ASDL process algebras | √ | × | × | √ | √ | × | √ | × |
| LOTOS/CADP | √ | × | × | √ | √ | × | √ | × |
| π -calcul | √ | × | × | √ | √ | × | √ | × |
| Temporal theories | √ | × | × | √ | √ | × | √ | × |
| Temporal Logic and ACTL Logic | √ | × | × | √ | √ | × | √ | × |
| Diapason approach | √ | × | × | √ | √ | × | √ | × |
| FIACRE languag e | √ | × | × | √ | √ | × | √ | × |
| MDA-UML-S | √ | × | × | √ | × (semi-formall) | × | √ | × |
| AAVF-IHM (Alexandre.Cor) | × | √ | √ | × | √ | √ | × | √ |
| CTT –B appraoch | √ | √ | √ | √ | √ | √ | √ | √ |

## 5. Conclusion

Several approaches have been presented in this paper, these approaches aimed at formalizing the behavior of web services. However, It should be noted that these works have been proposed, but none of them fully satisfies the requirements of developments of web applications:

The proposed works are centered around the architectural aspect (BPEL4WS) of development and aimed at formalizing the behavior of web services (Only the orchestration of web services gives a concrete vision that allows the expression of an executable process; for the majority, this is expressed through the language BPEL4WS) but without addressing the technological aspect (Ex: AJAX) of web applications. Although this aspect of technology has an essential effect on the process and the overall behavior of the application, including the partner web services themselves.

Even for the architectural aspect, these proposals remain partial solutions to the problems of the composition of the safe web services. The generated specifications are too abstract to be directly supported by an implementation language. These specifications correspond to the conceptual level of development considered. A step of refinement (coding) of these specifications becomes essential.

## References

Abrial, J. R. (1996). *The B-Book*. Combridge University Press. https://doi.org/10.1017/CBO9780511624162

Andrea, F. (n.d.). Web Services: a Process Algebra Approach.

Bolognesi, T., & Brinksma, E. (1987). Introduction to the ISO specification language LOTOS. *Computer Networks and ISDN systems*, *14*(1), 25-59. https://doi.org/10.1016/0169-7552(87)90085-7

CADP. (2020). *Construction and Analysis of Distributed Processes*.

Cau, A., Moszkowski, N., & Zedan, H. (2006). *Interval temporal logic*. Retrieved from http://www.cse.dmu.ac.uk/STRL/ITL/

Chirichiello, A., & Salaun, G. (2005, September). Encoding abstract descriptions into executable web services: Towards a formal development. In *The 2005 IEEE/WIC/ACM International Conference on Web Intelligence (WI'05)* (pp. 457-463). IEEE.

De Nicola, R., & Vaandrager, F. (1990, April). Action versus state based logics for transition systems. In *LITP Spring School on Theoretical Computer Science* (pp. 407-419). Springer, Berlin, Heidelberg. https://doi.org/10.1007/3-540-53479-2_17

DIS - Universita di Roma “La Sapienza”.

Dodani, M. H. (2004). From objects to services: A journey in search of component reuse nirvana. *Journal of Object Technology, 2004*. https://doi.org/10.5381/jot.2004.3.8.c5

Foster, H., Uchitel, S., Magee, J., & Kramer, J. (2003, October). Model-based verification of web service compositions. In *18th IEEE International Conference on Automated Software Engineering, 2003. Proceedings.* (pp. 152-161). IEEE.

Foster, H., Uchitel, S., Magee, J., & Kramer, J. (2005, July). Tool support for model-based engineering of web service compositions. In *IEEE International Conference on Web Services (ICWS'05)* (pp. 95-102). IEEE. https://doi.org/10.1109/ICWS.2005.119

Foster, H., Uchitel, S., Magee, J., & Kramer, J. (2006, May). LTSA-WS: a tool for model-based verification of web service compositions and choreography. In *Proceedings of the 28th international conference on Software engineering* (pp. 771-774). https://doi.org/10.1145/1134285.1134408

Gartner, (2018). Gartner Group: founded in 1979, is an American consulting and research company in the field of advanced techniques. It conducts research, provides consulting services, maintains various statistics and maintains a specialized news service.

Hamadi, R., & Benatallah, B. (2003, January). A Petri net-based model for web service composition. In *Proceedings of the 14th Australasian database conference-Volume 17* (pp. 191-200).

Kowalski, R., & Sergot, M. J. (1986). A logic-based calculus of events. *New generation computing, 4*(1), 67-95, 1986. https://doi.org/10.1093/ml/67.1.95

MacKenzie, C. M., Laskey, K., McCabe, F., Brown, P. F., Metz, R., & Hamilton, B. A. (2006). Reference model for service oriented architecture 1.0. *OASIS standard*, *12*(S 18).

Manna, Z., & Pnueli, A. (1990). A hierarchy of temporal properties. *Proceedings of the 9th ACM Symposium on Principles of Distributed Computing (PODC'90)*, 1990. https://doi.org/10.1145/93385.93442

Mateescu, R. (1998). *Vérification des propriétés temporelles des programmes paralleles* (Doctoral dissertation, Institut National Polytechnique de Grenoble-INPG).

Mateescu, R. (2003). *Logiques temporelles basées sur actions pour la vérification des systemes asynchrones*. https://doi.org/10.3166/tsi.22.461-495

Meftah, M. C., & Kazar, O. (2015). Web Applications Development by Formal Refinement Approach. *International Journal of Software Engineering and Its Applications*, *9*(12), 73-98. https://doi.org/10.14257/ijseia.2015.9.12.07

Milner, R. (1989). *Communication and concurrency* (Vol. 84). Englewood Cliffs: Prentice hall.

Milner, R. (1999). *Communicating and mobile systems: the pi calculus*. Cambridge university press.

MonInfo. (2005). *Le monde informatique n: 1139.*

Paternò, F., Santoro, C., & Spano, L. D. (2012). *ISTI-CNR W3C Working Group*.

Pnueli, A., & Manna, Z. (1992). The temporal logic of reactive and concurrent systems (Volume I: Specification). *Springer*, *16*, 12. https://doi.org/10.1007/978-1-4612-0931-7

Pourraz, F. (2007). *Diapason: une approche formelle et centrée architecture pour la composition évolutive de services Web* (Doctoral dissertation, PhD thesis, Université de Savoie, LISTIC).

Rouached, M., & Godart, C. (2006, December). Securing web service compositions: Formalizing authorization policies using event calculus. In *International Conference on Service-Oriented Computing* (pp. 440-446). Springer, Berlin, Heidelberg. https://doi.org/10.1007/11948148_37

Rouached, M., Gaaloul, W., Van Der Aalst, W. M., Bhiri, S., & Godart, C. (2006, October). Web service mining and verification of properties: An approach based on event calculus. In *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"* (pp. 408-425). Springer, Berlin, Heidelberg. https://doi.org/10.1007/11914853_72

Rouached, M., Perrin, P., & Godart, C. (2006b). Towards formal verification of web service composition. *4th International Conference on Business Process Management (BPM 2006)*. https://doi.org/10.1007/11841760_18

Salaün, G., Ferrara, A., & Chirichiello, A. (2004, September). Negotiation among web services using LOTOS/CADP. In *European conference on web services* (pp. 198-212). Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-30209-4_15

Software Tools for Designing Reliable Protocols and Systems http://cadp.inria.fr/ Version 2.816 last updated on 2020/03/10.

Solanki, M., Cau, A., & Zedan, H. (2006, May). Asdl: A wide spectrum language for designing web services. In *Proceedings of the 15th international conference on World Wide Web* (pp. 687-696). https://doi.org/10.1145/1135777.1135878

Tim, B. (2005). *In the opening statement of their conference* (Web 2.0 Conference 2005). September 30, 2005.

Via Salaria 113, 00198 Roma, Italia.

W3C. (2005). *Web Services Choreography Description Language Version 1.0 W3C Candidate Recommendation 9 November 2005*.