WS-Policy4MASC Version 0.8

Vladimir Tosic, Abdelkarim Erradi, Piyush Maheshwari
vladat@computer.org, aerradi@cse.unsw.edu.au, piyush@cse.unsw.edu.au
School of Computer Science and Engineering
The University of New South Wales
NSW 2052, Australia

UNSW-CSE-TR-0702
Technical Report, January 2007

THE UNIVERSITY OF
NEW SOUTH WALES

School of Computer Science and Engineering
The University of New South Wales
NSW 2052, Australia

Abstract

WS-Policy4MASC is a new XML language that we have developed for policy specification in the Manageable and Adaptable Service Compositions (MASC) middleware. It can be also used for other Web service middleware. It extends the Web Services Policy Framework (WS-Policy) by defining new types of policy assertions. Goal policy assertions specify requirements and guarantees (e.g., maximal response time) to be met in desired normal operation. They guide monitoring activities in MASC. Action policy assertions specify actions to be taken if certain conditions are met or not met (e.g., some guarantees were not satisfied). They guide adaptation and other control actions. Utility policy assertions specify monetary values assigned to particular situations (e.g., execution of some action). They can be used by MASC for billing and for selection between alternative action policy assertions. Meta-policy assertions can be used to specify which action policy assertions are alternative and which conflict resolution strategy (e.g., profit maximization) should be used. In addition to these 4 new types of policy assertions, WS-Policy4MASC enables specification of additional information that is necessary for run-time policy-driven management. This includes information about conditions when policy assertions are evaluated/ executed, parties performing this evaluation /execution, a party responsible for meeting a goal policy assertion, ontological meaning, monitored data items, states, state transitions, schedules, events, and various expressions.

This research report provides technical details about WS-Policy4MASC solutions. First, we summarize the need for the language. Then, we list the requirements we have identified for a policy language to support middleware for QoS-aware and adaptive Web service composition. Then, we explain and discuss many architectural decisions in the language. They are illustrated with diagrams (XmlSpy diagrams of XML Schemas and UML diagrams) and examples. The Appendices contain XML files of detailed examples and XML schemas of the WS-Policy4MASC language grammar.

Table of Contents

1. About This Document

Context:

This work is a part of the research project "Building Policy-Driven Middleware for QoS-Aware and Adaptive Web Services Composition" sponsored by the Australian Research Council (ARC) and Microsoft Australia through the ARC Linkage Project LP0453880.

Relationship between this document and our other publications:

This document is a technical report, not an academic paper. The purpose of this document is to provide information about technical details that are usually not published in academic papers due to space limitations. On the other hand, some aspects relevant for academic papers (e.g., comparisons with related work, frequent references to papers where used ideas were suggested/used) are not treated in detail here. Interested readers are advised to first read our academic papers that provide a summary and critical analysis of the WS-Policy4MASC language and/or the MASC (Manageable and Adaptable Service Compositions) middleware and only then study the technical details presented in this document. The list of our publications in this area can be found at: http://masc.web.cse.unsw.edu.au/.

History of this document:

Version and Date	***Major Characteristics of the Version***
0.1 2006/05/15	An initial draft in a notebook.
0.2 2006/05/31	The first typed draft of the requirements for and description of the WS-Policy4MASC language, as well as some solutions for major language constructs.
0.3 2006/06/08	Added structure, formalism, and explanations, particularly on the introductory part (motivation and requirements).
0.4 2006/07/25	Adds much more detail in the grammar for adaptation, further precision on the the requirements, and some brainstorming of architectural solutions. The first version that can be considered a draft of the language specification.
0.5 2006/08/02	Adds explanations of architectural solutions and a list for near-future tasks.
0.6 2006/09/25	Substantial new work on goal policy assertions, utility policy assertions, meta-policy assertions, as well as specification of states, events, and monitored data items. Significant simplifications in the approach to the WS-Policy4MASC language. In particular, the language schema is broken into a number of smaller schemas for different policy assertion types. The concept of a contract is postponed for future discussion.
0.7 2006/10/09	Some changes in the text are added to address some comments raised for the previous version. There are no changes in the language grammar, although many new TODO items related to these changes are identified.
0.8 2006/12/31	Substantial new work and significant redesign of the language. The main novelty are new specifications of meta-policy assertions and policy conflict resolution strategies.

Current level of maturity:

This document should be considered a mature draft of the WS-Policy4MASC language specification, but not the final language specification, because the design of the language and the corresponding MASC middleware is incremental and still on-going.

II) The Need for the WS-Policy4MASC Language

WS-Policy4MASC is developed as part of the project "Building Policy-Driven Middleware for QoS-Aware and Adaptive Web Services Composition". The goal of WS-Policy4MASC is formal, machine processable representation of policies for QoS (primarily performance and reliability) management and dynamic adaptation. It will be possible to use descriptions in this language as a declarative input into MASC middleware – based upon the contents of the WS-Policy4MASC policies (particularly: policy assertions), the MASC middleware will configure itself to provide QoS monitoring and dynamic adaptation of Web service compositions.

The term 'policy' is extremely ambiguous and means different things to different people. A general definition is that a policy is a declarative, high-level description of goals to be achieved and actions to be taken in different situations. However, this is ambiguous. (For example: How 'high'/abstract should a 'high-level' be?) Also, what general policy-based management literature calls 'policy' is called 'policy assertion' in WS-Policy, while what general policy-based management literature calls 'policy set' is called 'policy alternative' or 'policy' in WS-Policy (an ongoing standardization effort that is extended by our work). Throughout this document, we will use the terminology of WS-Policy.

In this project, WS-Policy4MASC policies can be occasionally read and written by humans, but they are intended for automatic processing by the MASC middleware. Consequently, the goal is to make these policies (particularly: their contained policy assertions) executable, not to make them high-level/abstract enough for use by non-technical humans (e.g., business managers). The language and the middleware will directly support only a few very high-level/abstract policy assertions and only those that are related to the dynamic adaptation and optimization of Web service compositions from the business value perspective. (The default algorithm will maximize business value, but we will also explore some policy-based customization of criteria used in making dynamic adaptation decisions.) One of the items for future work is exploring different layers of policy abstraction and, if appropriate, adding other types of high-level policy assertions. At this time, it seems that this is not appropriate with our project scope. We assume that there will be some future WS-Policy4MASC language tools that will enable relatively easy authoring of WS-Policy4MASC by people who have technical knowledge about Web service compositions and their execution, but who are not necessarily WS-Policy4MASC experts. Such tools are outside the scope of this project.

It must be clear that the design of a policy language is NOT the primary goal of this project. A new language by itself is not very useful. However, the design of an appropriate policy language is a necessary prerequisite for the development of the MASC middleware that is the primary goal of this project. In the same way that design of used data structures determines characteristics (e.g., speed) of algorithms, the design of management information representation critically determines characteristics (most importantly, functionality, but also performance) of management algorithms/protocols, as well as design of middleware that implements these algorithms/protocols.

It should be also clear that a design of an appropriate policy language is not a trivial task. While it is very easy to come up with a design f SOME language, it is quite hard to come up with design of a language that satisfies requirements for functionality, non-functional characteristics (e.g., simplicity, learnability, ...), as well as various tradeoffs (e.g., comprehensiveness vs. simplicity). This is an advanced, academic research project, so a particular care will be given to a study of benefits and limitations of various options, to substantiate the choices made in the language. One of the difficulties, but also benefits, of WS-Policy4MASC language development is in providing such a study.

For example, QoS is a very broad area, which can be classified into technical QoS, such as performance (e.g., response time) and availability, and business QoS, such as prices, penalties, and profit. There have been many specifications of technical QoS, but specification of business QoS is under-explored and requires additional research. Also, there are still research questions related to how to best support on the language level various dynamic (run-time) adaptation activities, so that middleware can automatically execute (policy-based) descriptions of these activities in an efficient way.

For these reasons, the design of the new policy language WS-Policy4MASC is a contribution of this project, which has to be achieved to realize the goals set up for the MASC middleware.

III) A Short Note on Verification and Validation

Verification and validation of the project were discussed in more details in previous documents discussing the overall project. However, due to the importance of this topic for the project success, a short summary is provided here.

It is important to show that the language is: 1) feasible, 2) useful, and 3) optimal (under some predefined criteria set). Feasibility will be proved through development of appropriate XML Schema grammar, proof-of-concept WS-Policy4MASC language tools, and the proof-of-concept MASC middleware using the language. The usefulness will be illustrated on example case studies. The optimality is the hardest to show. Due to the limited resources for this project (most importantly, time), the language need not be "perfect" (as there is no perfection), but arguments in favor of design choices must be given and some experimental studies of the benefits and limitations (e.g., performance overhead) of the adopted solutions should be performed.

IV) Major Requirements for a Policy Language to Support Middleware for QoS-Aware and Adaptive Web Service Composition and Corresponding High-Level Architectural Decisions in WS-Policy4MASC

We have determined the requirements for a policy language to support middleware for QoS-aware and adaptive Web service composition based on the document "Major Requirements for a System for Comprehensive Management of Mobile/Embedded XML Web Services" [TLT05] and motivational examples determined in this project.

The major requirements identified at this time are listed next. After every requirement, its motivation is briefly explained after the '$' sign. Then, importance/priority of this requirement for WS-Policy4MASC and/or corresponding high-level architectural decisions for WS-Policy4MASC are briefly discussed after the '=>' sign.

1. Formality, Precision, XML Encoding

The language must be formal, precise (without ambiguities), and encoded using XML Schema.
$ Formalism and precision of the language are necessary for machine processing without human intervention. The use of XML Schema is needed because Web services are built upon a set of XML-based technologies and because it has technical advantages over related technologies (particularly, Document Type Definitions – DTDs).
=> WS-Policy4MASC will satisfy this requirement.

2. Compatibility with Industry Standards for Web Services

The language SHOULD be compatible with Web service technologies widely accepted in industry, particularly with industry standards.
$ This is needed for easier acceptance of the language, as well as for potential reuse of tools.
=> WS-Policy4MASC will satisfy this requirement. First, WS-Policy4MASC will be compatible with WSDL and WS-BPEL, which are the most closely related and most widely used industry standards. Second, it will be an extension of WS-Policy [WSP04, WSP06]. While WS-Policy is not yet an industry standard (it was submitted to W3C in April 2006), it is supported by a number of key companies (including Microsoft). Since WS-Policy4MASC will contain many diverse concepts, this extension can be done in several ways. It would be beneficial to find such as an extension way that if somebody does not understand specific WS-Policy4MASC extensions, they can still understand standard WS-Policy tags. It is an open question whether such an extension can be found, but the current objective is to try to find it.

3. Compatibility with Microsoft Solutions for Web Services

The language SHOULD be compatible with Microsoft solutions in the Web services domain.
$ This is beneficial (if not required) because Microsoft is the sponsor of our project. However, this also has a research advantage – most research projects in the area of Web services are Java-based. Our experience is that Microsoft Web service solutions are different (sometimes subtly, sometimes considerably) and that these differences make Java-based solutions not always useful in the Microsoft world. On the other hand, Microsoft Web service solutions are widely used in practice. Providing solutions that are general and work for both Microsoft and Java-based technologies, or at least are proven to work with Microsoft technologies, increases originality of our research.
=> WS-Policy4MASC will satisfy this requirement. The major problem is that the current (pre-release) version of Microsoft (Windows) Workflow Foundation (WF) is not based on WS-BPEL, but on a significantly different format based on XAML (Extensible Application Markup Language). Our approach will be to first support in WS-Policy4MASC the basic process (workflow) concepts that appear in both WS-BPEL and XAML and then to provide extensions for specifics of both formats. Hereafter, we will use the term "process" instead of the term "workflow" and to denote Web service compositions supported by WS-BPEL and XAML.

4. Specification of Management Information for QoS Monitoring and Dynamic Adaptation

The language MUST enable specification of management information for both Web service QoS (performance) monitoring and dynamic (run-time) adaptation activities.

$ This is required by the scope of this particular research project. As detailed motivation was given in previous documents, only a brief summary is given here. QoS monitoring is necessary for determining the health of the monitored system to be able to control its behavior. Dynamic adaptation is an important form of behavior control. Both monitoring and control are needed to meet/surpass QoS guarantees, determine and fix performance problems and functional faults, perform accounting and billing, etc.
=> WS-Policy4MASC will support both Web service QoS monitoring and dynamic adaptation. However, support for dynamic adaptation will have higher priority, because the potential research impact is higher. (There are much more works on QoS monitoring for Web services than on dynamic adaptation.)

5. Language Modularity

The language MUST be modular, i.e., composed of several sub-formats for different aspects of Web service QoS monitoring and dynamic adaptation. These sub-formats SHOULD reuse a common general framework, concepts, and specification elements (e.g., specification of expressions).
$ In this research project, there is relatively significant uncertainty about the features of the resulting language. To handle this uncertainty, we have chosen incremental (and agile) development process for the the language – we will produce a limited working version fast, learn from it, and gradually grow it into a more complete language. Modularity of the language is necessary for such development. In addition, modularity enables that if the language is used in a limited way, only the corresponding modules have to be supported by the tools that process language specifications.
=> WS-Policy4MASC will satisfy this requirement, primarily by leveraging the modular nature of XML Schema specifications. The common general framework will be based on WS-Policy. The common concepts come from the theory of policy-based management--such as the [K&W04] concepts of action policies (assertions), goal policies (assertions), and utility policies (assertions)--will be reused throughout the language. The common specification elements (e.g., specification of expressions) will be mostly reused from the Web Service Offerings Language (WSOL) [TPP05].

6. Language Extensibility

The language MUST be extensible.
$ Extensibility of the language is necessary to achieve incremental development, as well as to accommodate requirements that will be discovered in the future (similar situations occur almost always in research projects).
=> WS-Policy4MASC will satisfy this requirement, primarily by leveraging the extensible nature of XML Schema specifications.

7. Support for QoS Monitoring

To support QoS monitoring, the language MUST enable specification of which QoS metrics (e.g., response time) are monitored or calculated, when/where/how this is done, and how the monitored/calculated values are exchanged between management parties. However, the definition of monitored/calculated QoS metrics SHOULD be outsourced to external reusable ontologies.
$ There are many different QoS metrics, such as response time, throughput, availability, and many others. There are usually no universally accepted definitions of these QoS metrics (e.g., 'response time' has at least 2 widely used definitions). QoS can be measured not only for providers, but also for requesters, and management third parties. Even for one particular party, it can be measured at different points, at different times, and in different ways (e.g., interception, probing, sniffing). Measured values can be exchanged between different parties in different ways (e.g., separate push/pull messages or piggybacking in existing messages). Therefore, QoS monitoring requires specifying which QoS metrics are monitored/calculated, when/where/how this is done, and how the values are exchanged. Definitions of QoS metrics can be done in policy files. However, outsourcing them to external ontologies improves reusability and flexibility.
=> WS-Policy4MASC will satisfy this requirement for both monitored and calculated QoS metrics. Note however that ontological definition will be specified only for use in comparisons of various Web services. In principle, it can be used as configuration data for monitoring/calculation, but this might be too complicated. Therefore, additional element 'ConfigurationData' will be specified. It is intended as a placeholder for specifying whatever configuration data the monitoring party needs. For example, for calculated data items, it could contain a WS-Policy4MASC expression describing how to calculate this value. This configuration data could include the name of the implementation module to be used for measurement or calculation. However, this brings implementation details into WS-Policy4MASC files, which should be avoided. Therefore, it is better to keep internal tables specifying which ontological

definitions (of QoS metrics or context properties) are monitored by which internal modules. It is likely that there will be a generic MASC middleware module for calculations, and probably also generic modules for capturing current time and manipulating counters. This will cover the most frequently used QoS metrics. Modules for additional QoS metrics and context properties (e.g., determining geographic coordinates using GPS hardware) are currently outside the scope of this version of the WS-Policy4MASC language and the MASC middleware.

8. Support for Context Sensitivity

Analogously to QoS metrics, the language COULD also enable specification of which context properties (e.g., absolute or relative geographic location) are monitored or calculated, when/where/how this is done, and how the monitored/calculated values are exchanged between management parties. The definition of monitored/calculated context properties SHOULD be outsourced to external reusable ontologies.
$ Execution context is particularly important for Web services executing in mobile/embedded environments. Not only that operation of the Web service can be context-dependent, but also Web service management can be context-dependent. As argued in [TLT06], there are some syntax and semantic similarities, but also differences, between specification of QoS metrics and context properties. Consequently, additional effort required for this support is not huge.
=> This requirement has very low priority for WS-Policy4MASC, because this project is not focused on specifics of mobile/embedded Web services. (Specification of context information for management of mobile/embedded Web services has been researched for the Web Service Offerings Language (WSOL) version 1.4 [TLT06].) If it is supported to some extent, this support will be analogous to the WS-Policy4MASC support for specification of QoS metrics.

9. Support for Declaration of Capabilities/Requirements

To support dynamic adaptation, the language COULD enable specification of declarative information about Web service technical capabilities and/or requirements (e.g., level of encryption used in exchanged SOAP messages).
$ Security and privacy policy assertions that are currently expressed with WS-Policy extensions are such types of declarative capabilities/requirements information. In some situations, it might be important to specify what type of dynamic adaptation should be performed if not all composed Web services satisfy technical requirements. This could be a way of handling policy conflicts.
=> This requirement has low priority for WS-Policy4MASC because security and privacy policy assertions are not the focus of this research project. However, it seems that it is possible to achieve this automatically through the compatibility with WS-Policy.

10. Support for Specification of QoS Conditions

To support QoS monitoring, the language MUST enable specification of QoS conditions (requirements and guarantees) that are evaluated, when/where/how this evaluation is done, what party is responsible for satisfaction of these conditions, and how the monitored/calculated values are exchanged between management parties.
$ Without specifying this information, it is not possible to automatically determine whether the monitored QoS values are within the expected boundaries or not.
=> WS-Policy4MASC will satisfy this requirement. In particular, QoS conditions will be specified in the form of goal policy assertions.

11. Support for Specification of Other Conditions

Analogously to QoS conditions, the language COULD also enable specification of other conditions (requirements and guarantees), such as functional constraints (particularly, preconditions and postconditions), context constraints, and access rights.
$ Such conditions are very useful for other Web service activities, such as fault management and security management. The format and detail of the information that has to be specified for them are quite similar to QoS conditions (probably even simpler), so the additional effort required for this support is not huge.
=> WS-Policy4MASC will provide general support in the form of goal policy assertions that will be able to specify various conditions. QoS conditions will be specified in detail. Some other types of conditions (but not all possible types) will be supported, if there is time. Conditions that constrain whether one of the parties has declared (using corresponding policy assertions) particular capabilities and/or requirements

might be supported, if there is time. This is not a high-priority task because specification of multiple types of constraints was researched in the Web Service Offerings Language (WSOL) [TPP05].

12. Support for Specification of Monetary Implications

To support both QoS monitoring and dynamic adaptation, the language SHOULD enable specification of monetary implications of using Web services and management parties and of meeting or not meeting specified conditions. This includes specification of subscription prices, pay-per-invocation prices, pay-per-volume prices, monetary penalties for not satisfying conditions, monetary penalties for remaining in undesirable state for some time, and similar information about prices, monetary penalties, and payment models.
$ This information is necessary for billing activities. Furthermore, it will be the basis for the business-driven dynamic adaptation activities in this project.
=> WS-Policy4MASC will support this requirement and provide specification of various types of price/penalty information. This will be done in the form of special utility policy assertions.

13. Support for Dynamic Adaptation

To support dynamic adaptation, the language MUST enable specification of how satisfaction or non-satisfaction of various conditions causes dynamic adaptation activities, particularly for dynamic customization, fault handling, and proactive optimization in Web service compositions.
$ Such information is the crucial link between monitoring and control (including dynamic adaptation) activities. Without it, dynamic adaptation is not possible.
=> WS-Policy4MASC will support this requirement through action policy assertions.

14. Specification of Health States

To support dynamic adaptation, the language SHOULD support specification of health states of a Web service or a Web service composition as an intermediate step between satisfaction of conditions and execution of dynamic adaptation actions. This means that there is a need for two separate types of specifications: 1) how satisfaction or non-satisfaction of various conditions causes change of health states; and 2) which dynamic adaptation actions should be performed in particular health states.
$ The concept of health states is useful because it decouples events from management actions. More than one event can map into a health state. Therefore, relating a management action to a health state (instead of a set of events) leads to more compact and flexible specifications.
=> While WS-Policy4MASC will support this requirement, it is not a top priority. Both mentioned specification types will be modeled through action policy assertions.

15. Specification of Static Customization of Web Service Compositions

The language COULD enable specification of static customization of Web service compositions (i.e., customization before the composition is running).
$ This can be useful to represent so called 'business exceptions'. In this approach, the main case is represented as a stand-alone Web service composition, while special cases ('business exceptions') are represented through static customization policy assertions on the main case (instead of separate stand-alone compositions). While this reduces performance (since it is needed to process policy assertions to determine the result composition), it provides a clear link between the main case and the special case and can be advantageous for maintenance reasons. For example, a change in the description of the main composition can automatically apply to the special cases (if the change is not affecting the customization policies).
=> Although static customization is not the focus of our project (and it probably does not carry significant research novelty), WS-Policy4MASC will support this requirement because static customization is a simpler case of dynamic customization. Using the approach “from simpler to more complex” we will first address static customization in the MASC infrastructure (and thus the WS-Policy4MASC language) and then proceed to explore to which extent these solutions can be applied to dynamic customization.

16. Specification of Dynamic Addition/Deletion/Replacement in Web Service Compositions

To support dynamic adaptation, the language MUST enable specification of simple and complex dynamic changes in the Web service composition, expressed through addition, deletion, and/or interface preserving

replacement of Web services or sub-compositions. ("Interface preserving" means that the replacement Web service or sub-composition provides the same interfaces as the replaced one.) These changes can range from one simple addition/deletion/replacement to complex cases where multiple such additions/deletions/replacements are ordered in sequence, parallelism, and/or choice structures.
$ Addition, deletion, and interface preserving replacement constitute the basic set of mechanisms for dynamic adaptation of Web service compositions. They are also the basis of static adaptation. With them, many types of adaptation can be specified. For example, replacing a Web service with another Web service with a different interface can be modeled if interface adaptors are also added.
=> WS-Policy4MASC will support this requirement. Note that the set {addition, deletion, interface preserving replacement} is redundant – replacement can be used instead of addition or deletion, while it can be itself modeled as a deletion-addition combination.

17. Specification of Dynamic Changes of Selection/Iteration Conditions in Web Service Compositions

To support dynamic adaptation, the language COULD enable specification of replacement of conditions in selection and iteration constructs of the Web service composition.
$ For example, it might be needed that some Web service is executed 3 times, instead of 2. Similarly, it might be needed that the condition for executing some operation is changed from "amount > = $1000" to "amount => $1005". Microsoft (Windows) Workflow Foundation (WF) supports such dynamic changes.
=> This requirement has a very low priority for WS-Policy4MASC because it is not clear how often the need for such changes arises in practice and because these changes can be modeled (in a not very efficient, but still effective way) using replacement of activities evaluating these conditions.

18. Specification of Invocation Retries

The language SHOULD enable specification of application-level invocation retries (including information such as a retry interval, maximum number of retries, timeout period, and whether the Web service notifies about its availability) as a means of dynamic adaptation.
$ This information is useful for handling temporary unavailability of Web services. For example, such temporary unavailability can occur relatively frequently with Web services executing in mobile devices. If third parties are used between the consumer and the unavailable provider Web service, the requests can be stored at the last available third party and forwarded to the provider Web service when it becomes available.
=> WS-Policy4MASC will support this requirement, although probably not in the early versions. While this project is not focused on specifics of mobile/embedded Web services (where the need for this language feature occurs most often), this language feature could be also useful in a considerable number of scenarios where Web services are neither mobile nor embedded. (By the way, specification of application-level invocation retries for management of mobile/embedded Web services has been researched for the Web Service Offerings Language (WSOL) version 1.4 [TTP06].)

19. Specification of Cache Validity

The language COULD enable specification of cache validity (such as cache expiration or duration and cache invalidation condition) to support caching of Web service messages as a means of dynamic adaptation to temporary disconnectionof Web services.
$ Caching could be useful as one of the means to address temporary disconnection of Web services. However, caching of Web service messages is much more complicated than caching of simple Web pages, because Web service messages can contain very dynamic (sometimes even unpredictably dynamic) contents.
=> This requirement has a very low priority for WS-Policy4MASC because this project is not focused on specifics of mobile/embedded Web services (where the need for this language feature occurs most often) and because of the complexities of supporting caching of Web service messages. (Specification of caching information for management of mobile/embedded Web services has been researched for the Web Service Offerings Language (WSOL) version 1.4 [TTP06].)

20. Support for Contracts

The language SHOULD support the concept of a contract – a comprehensive binding and enforceable agreement containing all information about QoS metrics, context properties, other declarative

information, QoS conditions, other conditions (e.g., functional, access rights, and context conditions), monetary implications, relationships between condition satisfaction and (dynamic and static) adaptation activities, health states, (dynamic and static) additions/deletions/replacements in the Web service composition, change of selection/iteration conditions in the Web service composition, application-level invocation retries information, caching information, change of contracts, and/or other information relevant for the operation and adaptation of the Web service or Web service composition. If contracts are supported, then it MUST be possible to specify a contract for an individual Web service and for the whole Web service composition, while it COULD be also possible to specify a contract for a sub-composition, i.e., a specific part of a Web service composition.
$ Contracts are beneficial because they group various information necessary for monitoring and control. By assuming that between a group of parties (e.g., a provider Web service and its consumer) there can be only 1 contract of a particular type used at one time, manipulation of which information (e.g., conditions) is valid becomes easier.
=> This is not a priority for WS-Policy4MASC. (Note that pecification and manipulation of contracts for management of Web services has been researched for the Web Service Offerings Language (WSOL) [TPP05].) WS-Policy4MASC could support this requirement by modeling a contract as a WS-Policy policy alternative (i.e., a 'policy set' in other policy languages) with some additional information (e.g., about validity). If this support is developed, it would be possible to attach the contract to an individual Web service (e.g., described in WSDL), to a Web service composition (e.g., described in WS-BPEL or Microsoft's XAML), and maybe also to a sub-composition, (e.g., described in WS-BPEL or Microsoft's XAML). Since this requires specifying information about the boundaries of the sub-composition, this is not a priority for WS-Policy4MASC.

21. Support for Different Contract Types

The language COULD support separation between different contract types, such as the separation between "regular behavior" contracts and "adaptation" contracts. Regular operation contracts contain information up to the states changes resulting from satisfaction or non-satisfaction of contractual conditions, while adaptation contracts contain information about what (dynamic or static) adaptation actions to perform if a particular contract (of any contract type) is used, the system is in a particular state, and some additional conditions (e.g., about context) are satisfied. At one particular time, a Web service or a Web service composition has assigned exactly one contract from each of these two types. Not all combinations should be valid. Therefore, an "overall" contract could be used to specify possible combinations of regular behavior contracts and adaptation contracts.
$ Such separation enables that an adaptation contract can be changed without modifying the regular behavior contract and vice versa.
=> This is not a priority for WS-Policy4MASC. (Note that pecification and manipulation of contracts for management of Web services has been researched for the Web Service Offerings Language (WSOL) [TPP05].) If such support is added, it will be possible for adaptation contracts to refer to regular behavior, adaptation, or overall contracts. The additional conditions in the adaptation contract clauses could be modeled with the <relevantIf> element that contains some Boolean expression.

22. Support for Dynamic Contract Manipulation

To support dynamic adaptation, the language SHOULD enable specification of change (manipulation) of contracts (of any contract type). In particular, the simplest case of dynamic switching from the current contract to another predefined contract SHOULD be supported, while more complex cases involving contract negotiation COULD be supported.
$ As argued in [TPP05], manipulation of predefined contracts is simpler, faster, and with less overhead than addition/deletion/replacement of Web services in the Web service composition. Therefore, it can be a useful complement to the latter mechanism. Dynamic contract negotiation is somewhere in between in terms of power and complexity.
=> This is not a priority for WS-Policy4MASC. (Note that pecification and manipulation of contracts for management of Web services has been researched for the Web Service Offerings Language (WSOL) [TPP05].) If such support is added, it will enable specification of contract switching as one possible dynamic adaptation mechanism. The support for contract negotiation will be rudimentary, if any.

23. Reusability Constructs

The language COULD support reusability constructs, such as inclusion, template instantiation, and (if

contracts are supported) inheritance hierarchies of contracts.
$ Reusability constructs are useful for easier development of specifications.
=> This requirement has a very low priority for WS-Policy4MASC because the expressive power of the language was not specified as one of the objectives of this project and because it requires time that we do not have (it would make the already late project even more late). An exception is inclusion – since WS-Policy already supports it through the PolicyReference element, it will be supported from early versions of WS-Policy4MASC.

24. Automatic Code Generation

The language MUST be defined in such as way that automatic generation of code (e.g., C# classes) from XML schema is easy and produces understandable results.
$ This feature is crucial for easier development of the middleware that uses the policy language.
=> WS-Policy4MASC will support this requirement. In particular, the XML Schema of WS-Policy4MASC will be defined in such a way so that the Microsoft .NET tool xsd produces C# classes that are easy to understand. The drawback is that this will increase complexity of the language.

This list will be updated during the incremental development of the WS-Policy4MASC language and the MASC middleware. The incremental development of the language will enable to design constructs that best address the needs identified during the development of this MASC middleware and best support solutions built into the MASC middleware.

V) Explanation and Discussion of Some Architectural Decisions and Future Tasks for WS-Policy4MASC and This Document

Summary of the main differentiators between our project and related works:

The compatibility with WS-Policy is one of the differentiators of our project from related work. It is discussed in more detail in subsequent paragraphs. Further, the WS-Policy4MASC language had to be compatible with industry standards WSDL and WS-BPEL, but since we work with Microsoft .NET 3.0, also with Microsoft's descriptions of Web service compositions in XAML. Therefore, compatibility with Microsoft technology is also one of the differentiators from similar research projects, which are almost exclusively Java-based. Another differentiator is emphasis on business-driven dynamic adaptation, supported by modeling of business value of execution and management of Web service compositions.

Justification for extending WS-Policy:

The WS-Policy4MASC language is encoded using XML Schema. It is an extension of WS-Policy [WSP04, WSP06], although it could be argued that this is more for political reasons (possible future standard, Microsoft project sponsorship) than for technical necessity. (The details of how WS-Policy4MASC extends WS-Policy will be provided in subsequent paragraphs.) WS-Policy is currently used almost exclusively for security, reliable messaging, and a few other management areas that are not the focus of our project, so there is novelty in our WS-Policy extensions. We have also looked at extending WS-Agreement [LDK04] together with (or instead of) WS-Policy, but we have found that they are mutually incompatible, because they define semantically similar concepts (e.g., grouping operators) in an incompatible way. Therefore, it seemed to us that we have to use either one or the other description framework. WS-Agreement is standardized by the Global Grid Forum (GGF) and accepted predominantly by the Grid computing community, while WS-Policy is being standardized by the World Wide Web Consortium (W3C) and is accepted by a broader Web service community. Our search of Internet Web pages (performed in early 2006) showed that references to WS-Policy outnumbered references to WS-Agreement about 6 to 1 in general literature and about 4 to 1 in academic literature. This is one of the indicators that WS-Policy is more widely known and used in practice, which probably means that there are bigger investments in WS-Policy than in WS-Agreement. Compatibility with WS-Policy enables leveraging the results of these investments and protecting companies that already use WS-Policy from drastically changing their support for Web services. Further, the goal of the research project within which the WS-Policy4MASC language is developed is to provide solutions for policy-based management of general Web services, while there are other research projects exploring extensions of WS-Agreement. Microsoft (the sponsor of our project) leads in the development of WS-Policy, while it is not involved in the development of WS-Agreement (which might mean that they do not plan to use it in the near future). Having in mind all these arguments, we have decided to research extensions of WS-Policy. Note that the WS-Policy specification is currently undergoing an evolution within the W3C standardization process [WSP06]. We have decided to work with the past stable version of the specification described in [WSP04], but we plan to align WS-Policy4MASC with new WS-Policy versions once they are stable.

Relationship to the determined requirements for a policy language: These architectural decisions are towards satisfying the requirements 2 “Compatibility with Industry Standards for Web Services” and 3 “Compatibility with Microsoft Solutions for Web Services”, as well as parts of the requirement 1 “Formality, Precision, XML Encoding”.

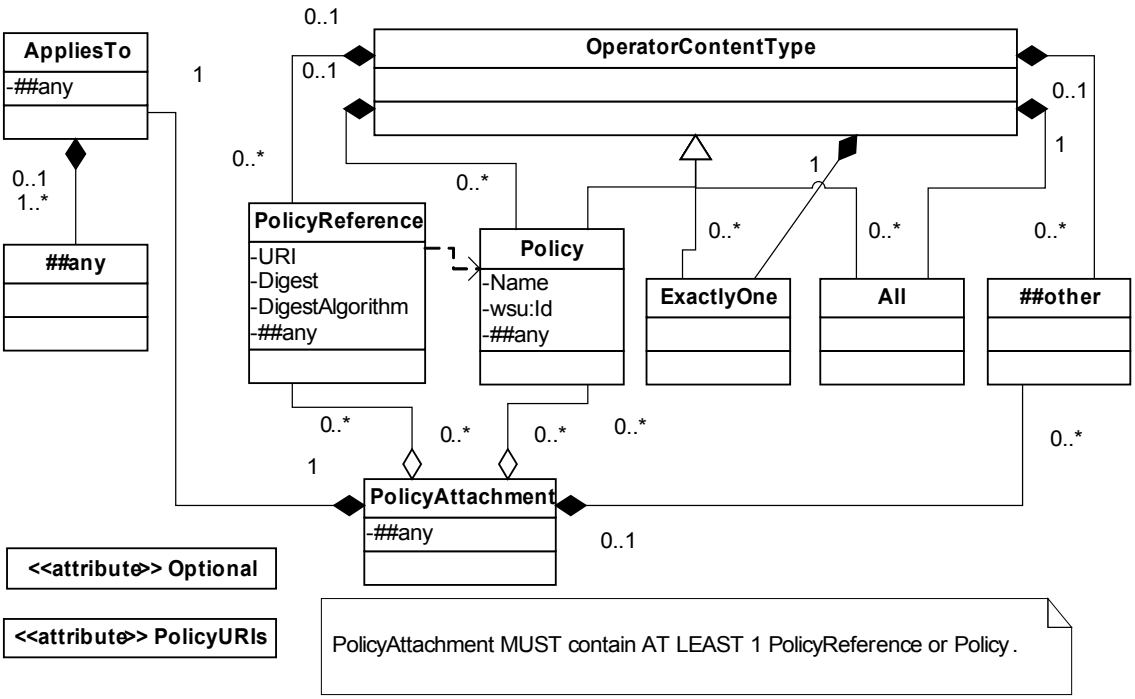

Figure 1. Relationships between the main WS-Policy concepts

Brief summary of the main WS-Policy concepts:

Figure 1 shows (in the UML notation) the elements and attributes defined in the WS-Policy language grammar (XML schema). In this document, only a brief summary will be provided. More complete and precise information can be found in [WSP04].

- While the basic concepts of WS-Policy are a policy assertion, a policy alternative, and a policy, the former two are not directly represented in the XML schema. A policy assertion can be any XML element conveying some useful information, so the "##other" placeholder is used in the XML grammar for WS-Policy. A policy alternative is a set of policy assertions, which can be expressed in WS-Policy with an <All> policy operator or a <Policy> element. A policy can be (but need not be) a choice between policy alternatives, which can be expressed in WS-Policy with the <ExactlyOne> policy operator within a <Policy> element.
- WS-Policy also enables policy attachment within the definition of concepts (e.g., WSDL constructs) to which a policy applies or in WS-Policy files outside these construct definitions. The first approach is to use extensibility elements in WSDL (or other Web services XML formats) to specify a WS-Policy <Policy> or <PolicyReference>. The second approach is to specify within WS-Policy files new <PolicyAttachment> elements that associate <Policy> or <PolicyReference> elements with their domains represented through the <AppliesTo> element.
- The WS-Policy XML schema also defines two attributes, "Optional" and "PolicyURIs" that can be used with different elements related to WS-Policy.

WS-Policy4MASC as an extension of WS-Policy:

WS-Policy4MASC is an extension of WS-Policy because it takes WS-Policy concepts and adds additional concepts that are compatible with the existing ones. Currently, all WS-Policy4MASC extensions of WS-Policy are modeled on the syntax level as WS-Policy policy assertions (and elements within them). However, they are semantically not "domain-specific policy assertions" because they are not for one particular domain – for example, one can model QoS, adaptation, security, functional constraints, billing information, and other information through the added WS-Policy4MASC concepts. In this version of WS-Policy4MASC, no extensions or changes to WS-Policy constructs (including <wsp:Policy>, <wsp:All>, and <wsp:ExactlyOne>) are done, so these constructs can be used with WS-Policy4MASC in exactly the same way as for any other WS-Policy policy assertions. (The various contract-related concepts from WS-Policy4MASC version 0.5 will probably be abandoned, but the final decision about this is postponed for the

future. If they are kept in the language, then they will probably have to be modeled as extensions of the <wsp:Policy> concept. This is discussed in more detail in a later paragraph.) WS-Policy4MASC policy assertions can refer to and be attached to WSDL constructs (e.g., Web services, endpoints/ports, operations, messages, message parts), and WS-BPEL/XAML constructs (e.g., process, sub-process, activity).

Relationship to the determined requirements for a policy language: These architectural decisions are towards satisfying the requirements 5 “Language Modularity” and 6 “Language Extensibility”. The support for policy attachment is towards satisfying requirements 2 “Compatibility with Industry Standards for Web Services”, 3 “Compatibility with Microsoft Solutions for Web Services”, and 9 “Support for Declaration of Capabilities/Requirements”. Note that if the contract-related concepts are added, they will be towards satisfaction of the requirements 20 “Support for Contracts”, 21 “Support for Different Contract Types”, and 22 “Support for Dynamic Contract Manipulation”. (At this time, these requirements are not satisfied.)

The meaning of policy alternatives in WS-Policy4MASC:

WS-Policy enables specification of policy alternatives. A policy alternative is a collection of policy assertions that are all relevant at the same time, but a system uses one and only one policy alternative at a time. Since WS-Policy4MASC is a WS-Policy extension, it supports this concept.

A question arises how to identify during run-time a policy alternative to be used. If supported policies of several parties (e.g., between a Web service and its requester) are considered, then the process of policy matching can be used to determine policy alternatives appropriate to all parties. However, even if policy matching is used, there might be more than one appropriate policy alternative, so additional explicit choice is needed. If WS-Policy4MASC file is in the WS-Policy normal form [WSP04], then every <wsp:All> element within the <wsp:ExactlyOne> element could be assigned a unique ID and the choice can be based on this ID. For example, if a Web service offers 4 policy alternatives “A1”, “A2”, “A3”, and “A4” and its client can support all of them, the client can do some internal analysis and send a message to the Web service stating that it wants to use “A3”. The problem is that the WS-Policy normal form is very verbose, so it not used by some (and maybe even many) policy authors.

Therefore, we have adopted the following approach. WS-Policy4MASC files written by humans need not be in the WS-Policy normal form and need not contain IDs for different policy alternatives. However, once the file is written, it MUST be translated into the WS-Policy normal form. (For example, this can be done by MASC middleware of exactly one party, to avoid that different normalization algorithms used by different parties produce different normal forms.) All relevant parties are distributed the same WS-Policy4MASC file in the normal form. The identification of policy alternatives is performed using XPath expressions over the contents of this file.

Note that it is sometimes needed to specify that in a particular situation (e.g., occurrence of a particular event), there are several action policy assertions that can be executed, but exactly one of them has to be chosen, based on some criteria. In WS-Policy4MASC, such situations MUST NOT be modeled using the WS-Policy <wsp:ExactlyOne> construct. Instead, all these action policy assertions must be specified in the same policy alternative, along with a meta-policy assertion describing which of them are conflicting and what criteria should be used for selection (more generally: for conflict resolution). This approach simplifies run-time operation of the MASC middleware, because selection of action policy assertions to execute is simpler than switching between policy alternatives produced by normalization when <wsp:ExactlyOne> is used. The concept of a policy alternative is not changed by this approach – the conflicting action policy assertions within one policy alternative are all relevant at the same time.

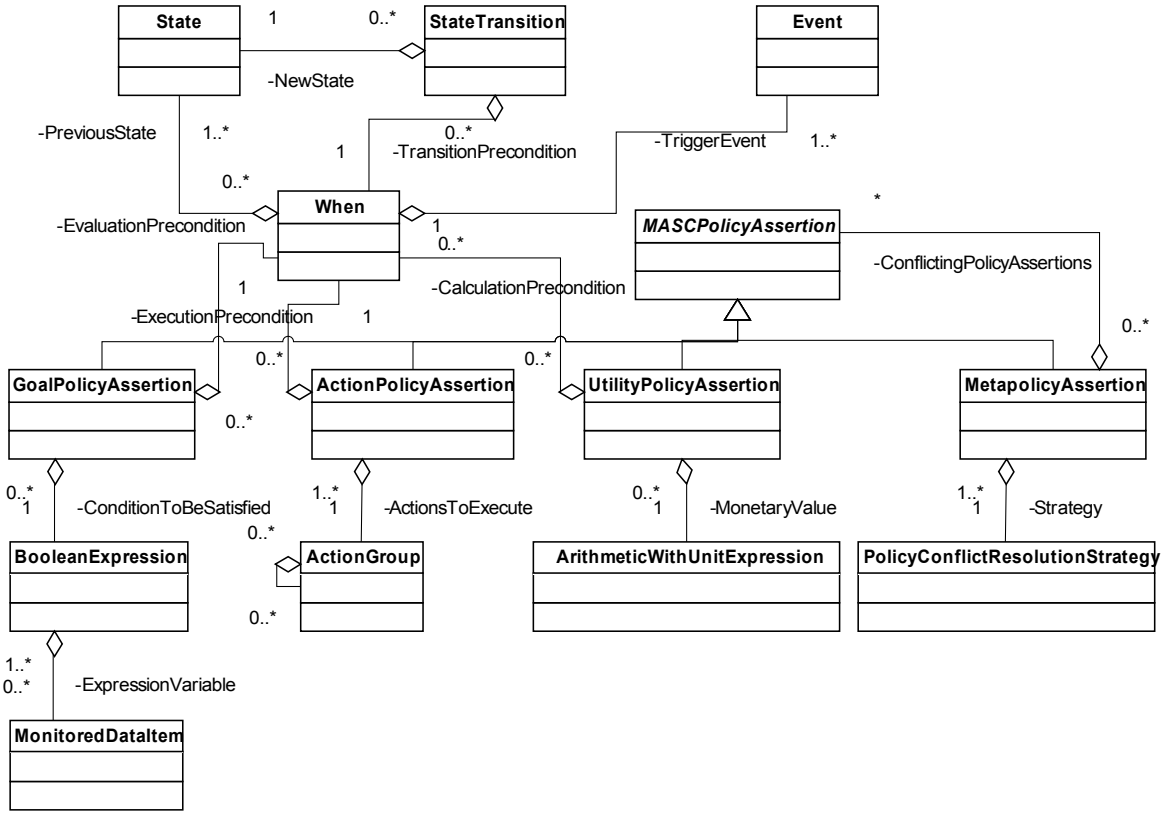

Figure 2. Some relationships between the main WS-Policy4MASC constructs

A high-level overview of WS-Policy4MASC concepts and constructs:

WS-Policy4MASC constructs can be classified into "real" management policy assertions and other, supporting constructs. In this paragraph we will provide only their high-level overview, while the details will be explained in subsequent paragraphs (sub-sections).

There are 4 types (categories) of **"real" management policy assertions**: goal policy assertions, action policy assertions, utility policy assertions, and meta-policy assertions. They specify information crucial for policy-driven management performed by the MASC middleware. Since these are the central concepts and constructs in the WS-Policy4MASC language, the majority of this section will be devoted to these constructs. For now, note that they all inherit from the the abstract policy assertion super-type MASCPolicyAssertion, which defines common attributes (e.g., which party performs evaluation/execution/calculation of a policy assertion). Elements and attributes defined in specific management policy assertions contain information specific to a particular type of a policy assertion (e.g., a goal policy assertion contains an attribute for specifying the stated goal).

The **supporting constructs** in WS-Policy4MASC are used to specify additional information that is necessary for run-time policy-driven management, but can be reused between various "real" management policy assertions. Examples of these supporting constructs are definitions of monitored data items, states, state transitions, schedules, events, scopes, and various expressions (Boolean, arithmetic, arithmetic with units, string, date/time/duration). Probably the most important of them is the <*When*> construct that specifies when something (e.g., evaluation of a goal policy assertion or execution of actions in an action policy assertion) should take place. It contains information about one or more states in which this can occur, one or more events (e.g., a Web service operation was executed) that can each individually trigger this occurrence, and an optional additional Boolean condition to be satisfied (it can be used for filtering). It is very important to note that these supporting constructs are also implemented as WS-Policy policy assertions from the syntax viewpoint, although they are not "real" management policy assertions from the semantic viewpoint. As will be elaborated below, this design decision was made to support reusability and ease automatic code generation. All these supporting constructs, along with MASCPolicyAssertion, inherit from an abstract super-type MASCConstruct, which defines one common attribute – a unique ID.

As mentioned above (and as will be revised later, after all WS-Policy4MASC constructs are elaborated), simplicity of automatic code generation from XML schemas into C# classes and reusability of specification elements had significant influences on the design of the WS-Policy4MASC language. For example, we

avoided the <*choice*> element in XML schemas for WS-Policy4MASC because it resulted in C# classes that were difficult to understand and handle. Instead, we decided to model potentially reusable supporting constructs as WS-Policy policy assertions and then reference them within the other WS-Policy4MASC constructs. These **references** are specified through instances of the element type MASCRef, which contains one attribute – ID of the referenced MASC construct. For every WS-Policy4MASC construct type, an instance of this element type is defined. (At present, references to different types of WS-Policy4MASC constructs are always defined as instances of this type. A future version of the WS-Policy4MASC language might contain a hierarchy of references parallel to the hierarchy of the referenced MASC constructs. This would help with semantic checks.) A drawback of our approach is that our solutions lead to specifications that are verbose. However, we hope that, in the future, WS-Policy4MASC files will be generated by graphical tools, so this verbosity of WS-Policy4MASC files will not be a strain for humans.

Some of the relationships between the main WS-Policy4MASC concepts are shown (in the UML notation) in **Figure 2**. While the details will be discussed in subsequent paragraphs (sub-sections), the figure is presented here to support an early general overview. The figure does not show all existing concepts and relationships – it is focused only on the main constructs (e.g., the 4 types of the "real" management policy assertions and a few main supporting constructs). In particular, the figure does not show that all shown constructs inherit from MASCConstruct. It also does not show that association is is implemented through instances of the MASCRef element type.

Relationship to the determined requirements for a policy language: The discussed design decisions support a number of identified requirements, such as 4 "Specification of Management Information for QoS Monitoring and Dynamic Adaptation", 5 "Language Modularity", 23 "Reusability Constructs", and 24 "Automatic Code Generation". A number of other requirements are satisfied through the support for individual types of management policy assertions and supporting constructs, as will be discussed later.

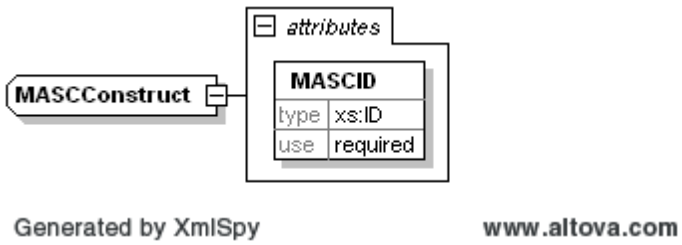

Figure 3. The structure of the MASCConstruct type

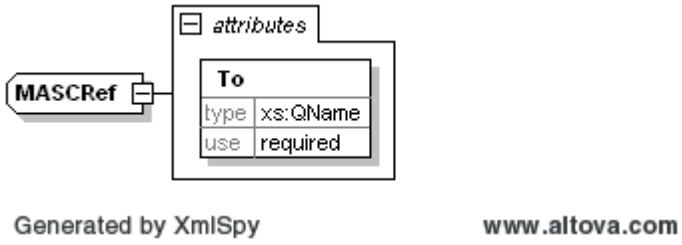

Figure 4. The structure of the MASCRef type

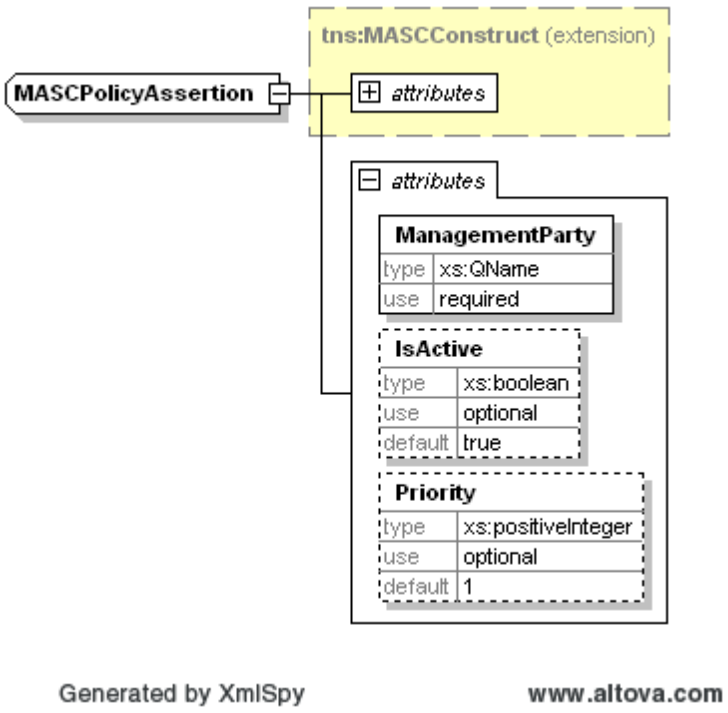

Figure 5. The structure of the MASCPolicyAssertion type

MASCConstruct, MASCRef, and MASCPolicyAssertion:

MASCConstruct, MASCRef, and MASCPolicyAssertion are the basic types in WS-Policy4MASC. While MASCConstruct and MASCPolicyAssertion are intended to be used as abstract super-types, many instances of the MASCRef type are defined.

MASCConstruct is the abstract super-type of all WS-Policy4MASC constructs that are intended to be used as WS-Policy policy assertions. **Figure 3** shows its structure (in the Altova XmlSpy notation). It contains 1 required attribute "MASCID" of the XML Schema data type ID. This means that every instance of a WS-Policy4MASC construct that inherits from MASCConstruct must have policy assertion must have a unique ID. Notice that this attribute is not the same as the WS-Policy attribute "wsp:Name" or the WS-SecurityPolicy attribute "wsu:Id". The latter two attributes are used for identification of complete policies, while "MASCID" is used for identification of individual policy assertions (and other MASC constructs). It seems that "wsp:Name" and "wsu:Id" can be for policy assertions, but this could lead to semantic errors. Another advantage is that WS-Policy4MASC processing tools can perform additional semantic checks in frequent situations when only WS-Policy4MASC policy assertions can be used, while other WS-Policy policy assertions are inappropriate. Examples of use of sub-types of MASCConstruct will be given later in this document.

MASCRef is the general type of references in WS-Policy4MASC. Many elements are defined as instances of this type. **Figure 4** shows its structure (in the Altova XmlSpy notation). It contains 1 required attribute "To", which contains some namespace-qualified MASCID.

MASCPolicyAssertion is the abstract super-type of all "real" management policy assertions. **Figure 5** shows its structure (in the Altova XmlSpy notation). It extends MASCConstruct and adds 3 optional attributes. The attribute "ManagementParty" of the XML Schema data type anyURI contains name of the party that performs evaluation of a goal policy assertion, execution of an action policy assertion, accounting of a utility policy assertion, or decision using a meta-policy assertion. The default value is the constant "MASC_WSORCHESTRATOR", which refers to the Web service orchestrator. The Boolean attribute "IsActive" specifies whether this management policy assertion is currently active. By default, it is. The positive integer attribute "Priority" contains priority (by default: 1) of this management policy assertion. The higher this number, the higher the priority is. Note that in the current version of the WS-Policy4MASC and MASC middleware priorities are not used, but this attribute is a placeholder for possible future work. Contrary to works that use priorities for decision which action policy assertion to execute when several are triggered simultaneously, our work uses business value (specified in utility policy assertions and calculated according to meta-policy assertions). In principle, our work on business value can be compatible with priorities, so that they are both used for decision making. However, our idea is to first make decisions based

on business value and only if there are several alternatives with equal business value then use priorities (if any is specified).

The definition of these 3 element types (and the <MASCPolicyAssertionRef) is in the file ws-policy4masc-pa.xsd , which is listed in **Appendix 4**. Since MASCConstruct and MASCPolicyAssertion are abstract types, we cannot provider their direct examples (except examples of their sub-types, which we will present later in this document). Examples of instances of MASCRef will also be given later.

Overview of the 4 supported types of "real" management policy assertions:

Currently, there are 4 types (categories) of policy assertions supported in WS-Policy4MASC:

1. **Goal policy assertions** specify requirements (e.g., 128 bit security must be used) and guarantees to be met (e.g., response time from a particular Web service operation/sub-process/process will always be less or equal to 2 seconds) in desired (normal) operation. In the MASC middleware, they guide monitoring activities (primarily: evaluation of various monitored conditions). They are represented with the GoalPolicyAssertion construct.
2. **Action policy assertions** specify actions to be taken if certain monitored conditions are met (e.g., some guarantees were not satisfied). For example, these actions can be removal, addition, replacement, skipping, or retrying of a sub-process (or individual activity) or process termination. In MASC, they guide adaptation and other control actions and a few aspects of monitoring (such as monitored data collection and/or exchange). All customization (versioning), correction (fault management), optimization (performance management), and other types of adaptation are specified in WS-Policy4MASC through action policy assertions. Action policy assertions are represented with the ActionPolicyAssertion construct.
3. **Utility policy assertions** specify monetary amounts (representing various tangible and intangible business values) assigned to particular run-time situations (e.g., satisfaction or non-satisfaction of some goal, execution of some action, entering a state, exiting a state, staying in a state for some time, etc.). They can be used by MASC for accounting/billing and, indirectly, for selection between alternative action policy assertions. They are represented with the UtilityPolicyAssertion construct.
4. **Meta-policy assertions** can be used to specify which action policy assertions are alternative (i.e., their conditions are satisfied at the same time) and which conflict resolution strategy should be used to decide which action policy assertion to execute. They are represented with the MetaPolicyAssertion construct.

All 4 supported types of policy assertions inherit from the MASCPolicyAssertion construct and will be discussed in more detail in subsequent paragraphs (sub-sections). Some relationships between these constructs (and, thus, their similarities and differences) were shown in the UML diagram in **Figure 2**. Goal policy assertions, action policy assertions, utility policy assertions and meta-policy assertions are subtypes of the abstract policy assertion element type MASCPolicyAssertion that defines common attributes. The 4 policy assertion types have additional attributes and child elements. The first 3 types of policy assertions (goal, action, utility) reference a <*When*> element describing in which state(s) and on occurrence of which event(s) a policy assertion should be processed. A goal policy references a Boolean expression with the condition to be evaluated – only if the given expression evaluates to "true" the goal was achieved/met/satisfied. An action policy assertion references a group of actions that are to be executed. A utility policy assertion contains an arithmetic with (currency) unit expression that determines associated monetary value. A meta-policy assertion does not reference a <*When*> element, but a set of mutually conflicting policy assertions. It contains information about a conflict resolution strategy (this will be explained in Subsection 3.4).

Relationship to the determined requirements for a policy language: The choice to support various types of policy assertions is towards satisfying the requirement 4 "Specification of Management Information for QoS Monitoring and Dynamic Adaptation". A number of other requirements are satisfied through the support for individual types of policy assertions, as will be discussed later.

WS-Policy4MASC policy assertion types vs. Kephart & Walsh policy types:

It should be noted that the WS-Policy4MASC use of the terms "goal policy assertion" and "utility policy assertion" is somewhat different than the terms "goal policy" and "utility policy" used by Kephart & Walsh [K&W04].

A WS-Policy4MASC goal policy assertion specifies a condition (requirement or guarantee) to be

achieved/met/satisfied, while a Kephart & Walsh goal policy specifies a desired state (situation) of the system. An individual WS-Policy4MASC goal policy assertion does not denote a complete desired state of the system, but a combination of all WS-Policy4MASC goal policy assertions valid at the same time determines a desired state of the system. This difference could be viewed as consistent with the WS-Policy distinction between the terms "policy" and "policy assertion".

An analogous situation is with the WS-Policy4MASC utility policy assertions. While for Kephart & Walsh a utility policy specifies utility value associated to a particular state (situation) of the system, in WS-Policy4MASC it a utility can be associated not only to staying in a state for some time, but also to state transitions, execution/non-execution of action policy assertions, satisfaction/non-satisfaction of goal policy assertions, etc. An individual WS-Policy4MASC utility policy assertion does not denote a complete utility value associated to a particular system, but a combination of all WS-Policy4MASC utility policy assertions valid at the same time determines a complete utility value.

On the contrary, the meaning of the WS-Policy4MASC term "action policy assertion" can be considered the same as the meaning of the Kephart & Walsh term "action policy". They both represent a set of actions to perform if particular conditions are met.

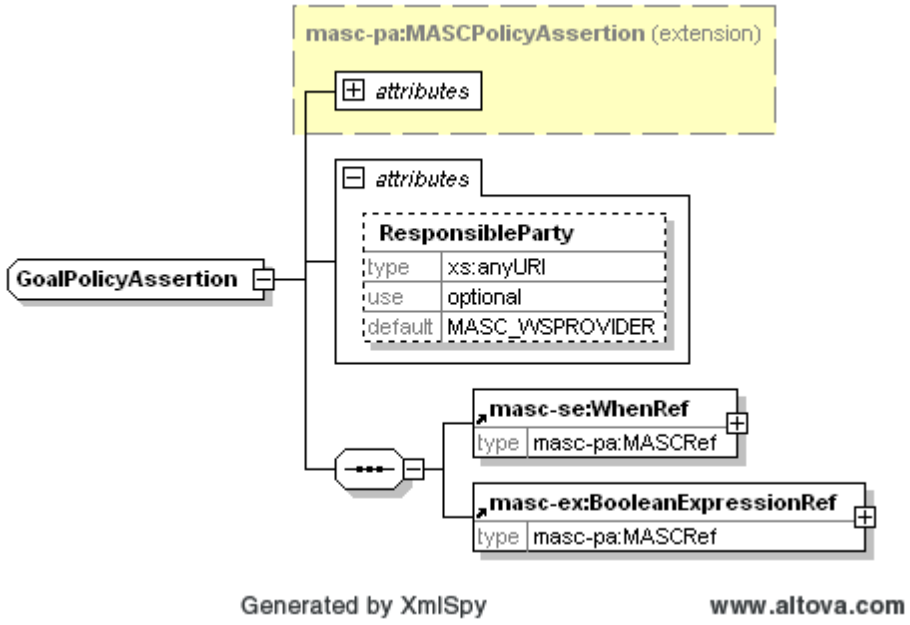

Figure 6. The structure of the GoalPolicyAssertion type

```
<masc-gp:GoalPolicyAssertion MASCID="ResponseTimeLimit" ResponsibleParty="MASC_WSPROVIDER" Man-
agementParty="MASC_WSPROVIDER">
 <masc-se:WhenRef To="tns:Executing-ReplyReceived"/>
 <masc-ex:BooleanExpressionRef To="tns:UpperLimitOfResponseTime"/>
</masc-gp:GoalPolicyAssertion>
```

Figure 7. An example of a WS-Policy4MASC goal policy assertion

Goal policy assertions:

A **goal policy assertion** specifies a condition (requirement or guarantee) to be achieved/met/satisfied in certain circumstances. It is specified in the **<GoalPolicyAssertion> element**, which is an instance of the GoalPolicyAssertion element type. There is some similarity between this WS-Policy4MASC concept ant the WSOL [TPP05] concept of a constraint.

Figure 6 shows (in the Altova XmlSpy notation) the structure of the **GoalPolicyAssertion element type**. This element type extends MASCPolicyAssertion and adds 2 required child elements and 1 optional attribute. The first child element is a MASCRef reference to a <*When*> element that specifies when the evaluation whether this condition was satisfied should occur. The second child element is a MASCRef reference to a <*BooleanExpression*> element that contains the condition to be evaluated – only if the given

expression evaluates to “true” the condition was achieved/met/satisfied. The optional attribute “ResponsibleParty” of the XML Schema data type anyURI contains information about which party (e.g., the provider, the requester, the orchestrator, a management third party, …) is responsible for meeting this condition. The default value of this attribute is “MASC_WSPROVIDER”, which refers to the provider Web service to which the goal policy assertion is attached. While provider Web services are usually responsible for meeting goal policy assertion conditions, some goal policy assertions have other responsible parties. For example, requesters are responsible for meeting functional pre-conditions.

The definition of this element type (along with the elements <*GoalPolicyAssertion*> and <*GoalPolicyAssertionRef*>) is in the file ws-policy4masc-gp.xsd , which is listed in **Appendix 5**.

The structures of the When element type and the BooleanExpression elemement type will be presented later in this document.

An example of a goal policy assertion is given in **Figure 7**. It is a part of the bigger case study given in **Appendices 1, 2, and 3**. In particular, this XML snippet is from **Appendix 2**. The example in Figure 7 states that the MASCID is “ResponseTimeLimit”, that the responsible party is the provider Web service and that the management party evaluating this goal policy assertion is also the provider. Further, it refers to the <When> construct “Executing-ReplyReceived” and <BooleanExpression> construct “UpperLimitOfResponseTime”. Since the <When> and <BooleanExpression> WS-Policy4MASC elements are explained later in this document, their examples will be given later.

The architecture of the MASC middleware contains general-purpose modules actually evaluating Boolean expressions specified in WS-Policy4MASC goal policy assertions (and other constructs, such as the <*When*> construct). While these modules are designed in the overall MASC middleware architecture, their full prototype implementation is not a priority for our research, because similar solutions have appeared in a number of recent Web service middleware architectures, including wsBus and WSOI. Due to the low research novelty of such modules, we will focus first on more challenging solutions for business-driven dynamic adaptation.

Relationship to the determined requirements for a policy language: These solutions are towards satisfying the requirements 7 “Support for QoS Monitoring”, 10 “Support for Specification of QoS Conditions”, and 11 “Support for Specification of Other Conditions”. They also leave the possibility for future extensions to satisfy the requirement 8 “Support for Context Sensitivity”.

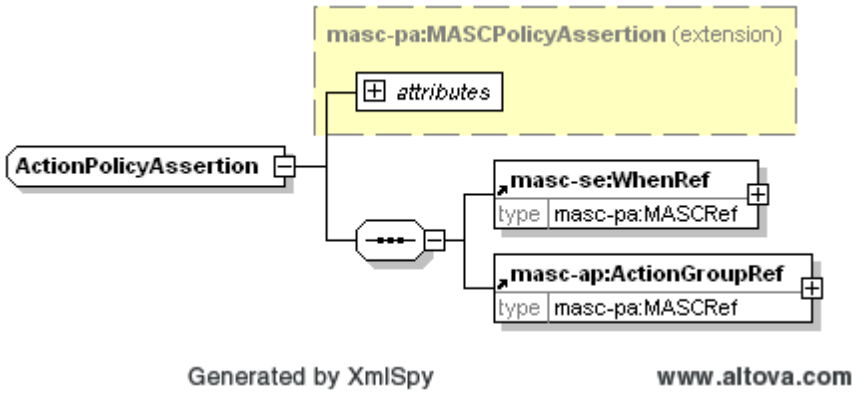

Figure 8. The structure of the ActionPolicyAssertion type

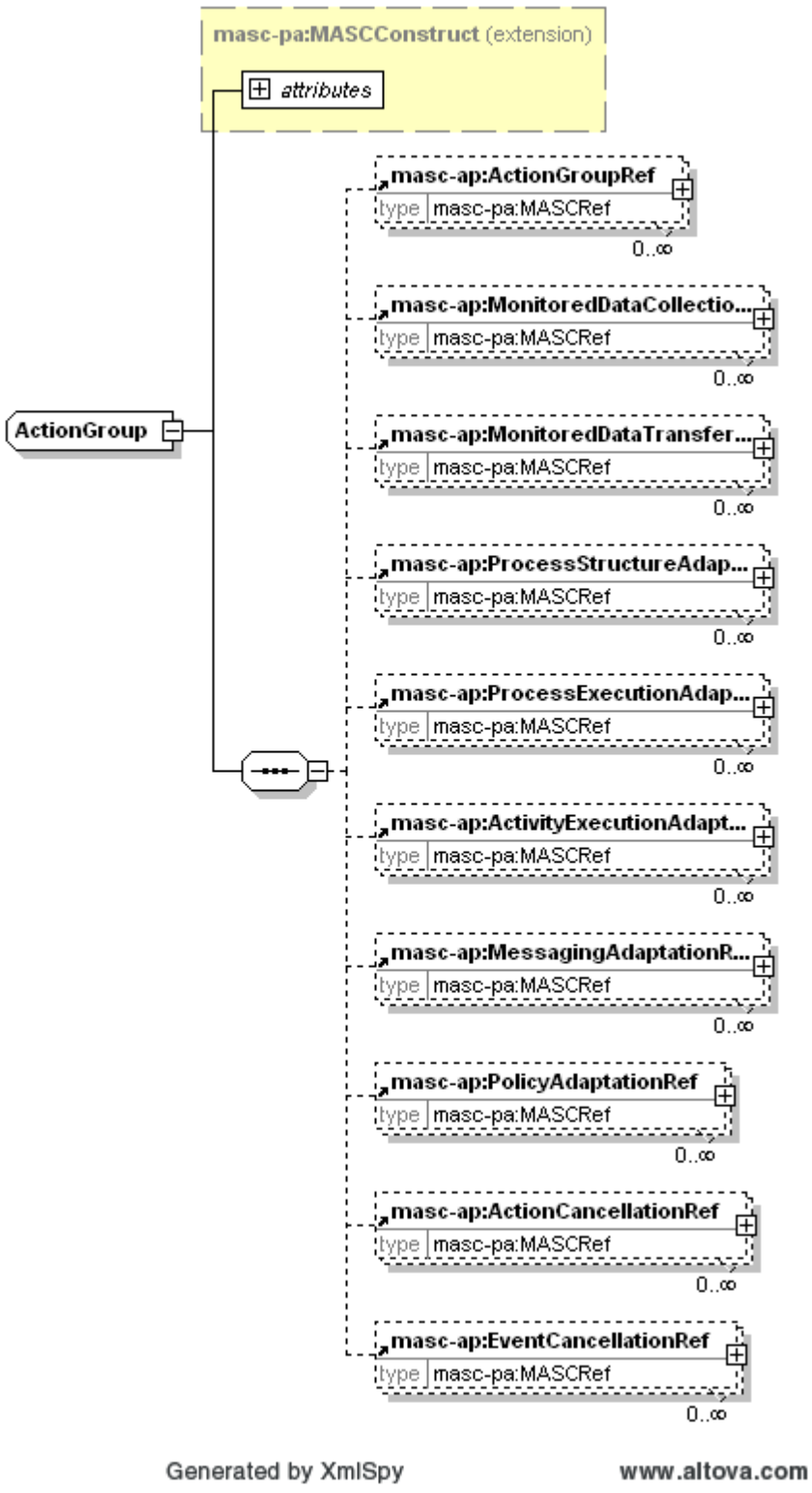

Figure 9. The structure of the ActionGroup type

Action policy assertions and an overview of supported types of actions:

An **action policy assertion** specifies a list of actions to be executed in certain circumstances. It is specified in the **<ActionPolicyAssertion> element**, which is an instance of the ActionPolicyAssertion element type.

Figure 8 shows (in the Altova XmlSpy notation) the structure of the **ActionPolicyAssertion element type**. This element type extends MASCPolicyAssertion and adds 2 required child elements. The first child element is a MASCRef reference to a <*When*> element that specifies when the execution of the listed actions should be performed. The second child element is a MASCRef reference to an <*ActionGroup*> element (supporting construct in WS-Policy4MASC) that contains the list of actions to be executed. We have decided to require

that actions must be grouped into action groups to increase reusability. Unfortunately, this leads to more verbose specifications.

The definition of this element type (along with the elements <*ActionPolicyAssertion*> and <*ActionPolicyAssertionRef*>) is in the file ws-policy4masc-ap.xsd , which is listed in **Appendix 6**.

Figure 9 shows (in the Altova XmlSpy notation) the structure of the **ActionGroup element type**. This element type extends MASCConstruct and adds 0 to many child elements that are MASCRef references to elements (supporting constructs in WS-Policy4MASC) of the types ActionGroup, MonitoredDataCollection, MonitoredDataTransfer, ProcessStructureAdaptation, ProcessExecutionAdaptation, ActivityExecutionAdaptation, MessagingAdaptation, PolicyAdaptation, ActionCancellation, and/or EventCancellation. Apart from ActionGroup (which enables nesting of action groups), the other listed element types describe various types of action that can be performed by the MASC middleware.

The definition of this element type (along with the elements <*ActionGroup*> and <*ActionGroupRef*>) is in the file ws-policy4masc-ap.xsd , which is listed in **Appendix 6**.

Examples of action policy assertions and action groups will be given bellow, after the different types of supported actions are explained.

Relationship to the determined requirements for a policy language: The solutions for action policy assertions (and various types of supported actions) are towards satisfying a number of the identified requirements. The details will be given for particular types of supported actions.

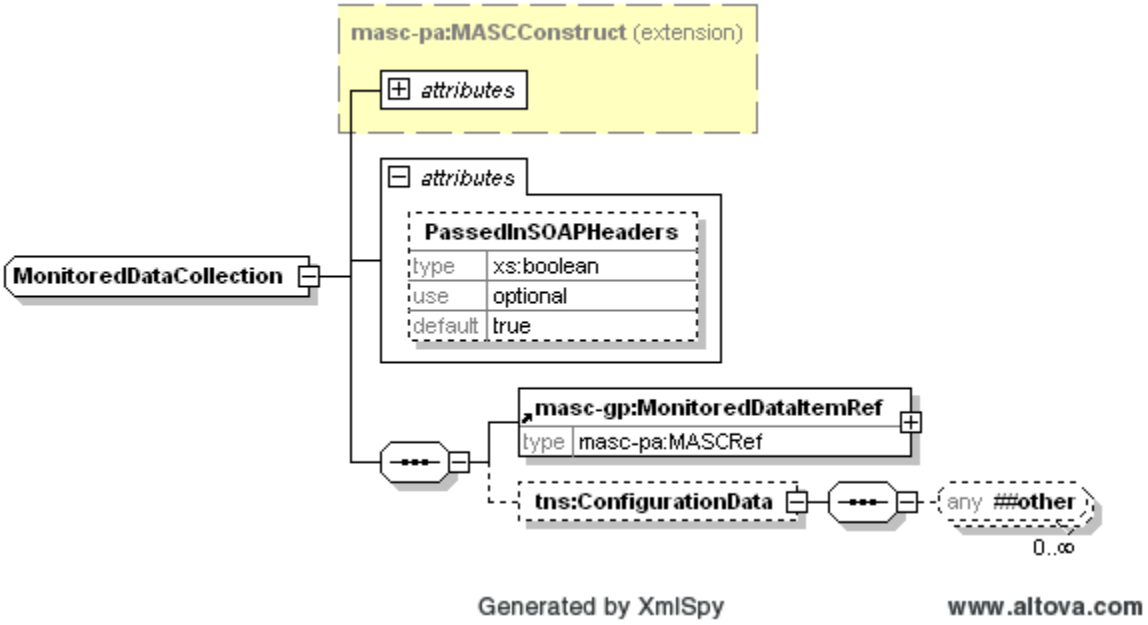

Figure 10. The structure of the MonitoredDataCollection type

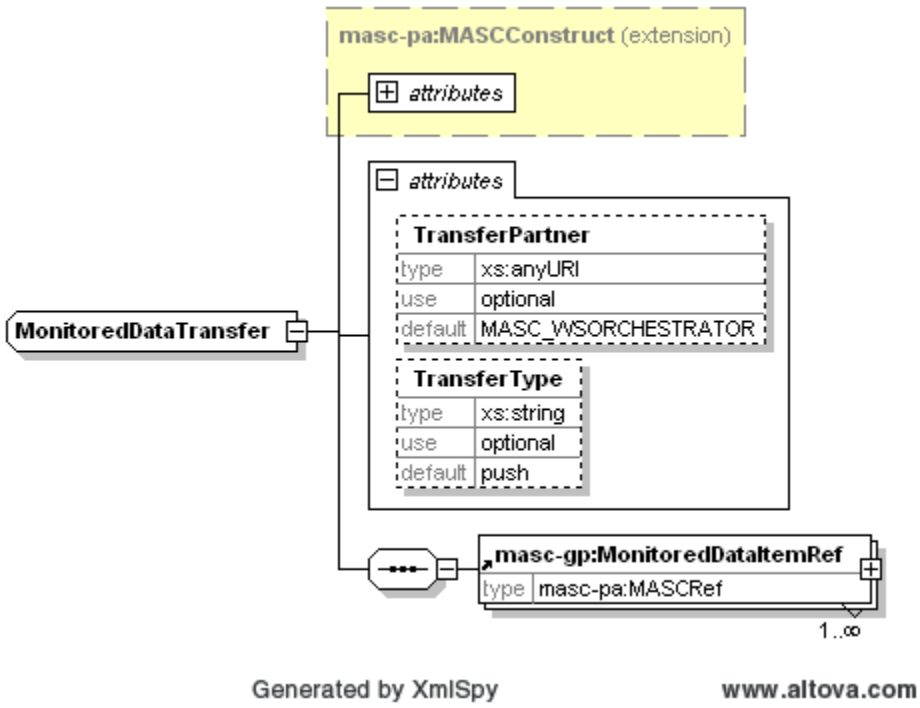

Figure 11. The structure of the MonitoredDataTransfer type

```
<!-- Definitions of monitoring data collection actions. -->
<masc-ap:MonitoredDataCollection MASCID="MonitoringOfResponseTimeStart-InMilliseconds">
 <masc-gp:MonitoredDataItemRef To="tns:ResponseTimeStart-InMilliseconds"/>
</masc-ap:MonitoredDataCollection>
<masc-ap:MonitoredDataCollection MASCID="MonitoringOfResponseTimeStop-InMilliseconds">
 <masc-gp:MonitoredDataItemRef To="tns:ResponseTimeStop-InMilliseconds"/>
</masc-ap:MonitoredDataCollection>
<masc-ap:MonitoredDataCollection MASCID="MonitoringOfResponseTime-InMilliseconds">
 <masc-gp:MonitoredDataItemRef To="tns:ResponseTime-InMilliseconds"/>
 <masc-ap:ConfigurationData>
  <configData:Start>tns:ResponseTimeStart-InMilliseconds</configData:Start>
  <configData:Stop>tns:ResponseTimeStop-InMilliseconds</configData:Stop>
 </masc-ap:ConfigurationData>
</masc-ap:MonitoredDataCollection>
<!-- Definitions of monitoring action groups. -->
<masc-ap:ActionGroup MASCID="MonitoringResponseTimeStart">
 <masc-ap:MonitoredDataCollectionRef To="tns:MonitoringOfResponseTimeStart-InMilliseconds"/>
</masc-ap:ActionGroup>
<masc-ap:ActionGroup MASCID="MonitoringResponseTimeStop">
 <masc-ap:MonitoredDataCollectionRef To="tns:MonitoringOfResponseTimeStop-InMilliseconds"/>
 <masc-ap:MonitoredDataCollectionRef To="tns:MonitoringOfResponseTime-InMilliseconds"/>
</masc-ap:ActionGroup>
<!-- Definitions of action policy assertions configuring monitoring. -->
 <masc-ap:ActionPolicyAssertion MASCID="MonitorResponseTimeStart"
ManagementParty="MASC_WSREQUESTER">
  <masc-se:WhenRef To="tns:Executing-RequestToBeSent"/>
  <masc-ap:ActionGroupRef To="tns:MonitoringResponseTimeStart"/>
 </masc-ap:ActionPolicyAssertion>
 <masc-ap:ActionPolicyAssertion MASCID="MonitorResponseTimeStop"
ManagementParty="MASC_WSREQUESTER">
  <masc-se:WhenRef To="tns:Executing-ReplyReceived"/>
  <masc-ap:ActionGroupRef To="tns:MonitoringResponseTimeStop"/>
 </masc-ap:ActionPolicyAssertion>
```

Figure 12. Examples of a WS-Policy4MASC goal policy assertion, action groups, and monitored data collection actions

MonitoredDataCollection and MonitoredDataTransfer:

The MonitoredDataCollection and MonitoredDataTransfer actions are used for configuration of how the monitored data items (SOAP message parts and/or QoS metrics) are collected by the MASC middleware and transferred between various management parties using the MASC middleware. While the main aspect of MASC monitoring, the evaluation of conditions, is described in WS-Policy4MASC goal policy assertions, these two actions are listed within WS-Policy4MASC action policy policy assertions. In earlier versions of WS-Policy4MASC these actions were described in separate WS-Policy4MASC supporting constructs, but we have noticed the syntax (and, to some extent, semantic) similarity with action policy assertions and decided to use action policy assertions.

Figure 10 shows (in the Altova XmlSpy notation) the structure of the **MonitoredDataCollection element type**. This element type extends MASCConstruct and adds 1 required child element, 1 optional child element, and 1 optional attribute. The first, required, child element is a MASCRef reference to a <*MonitoredDataItem*> element that describes the message part or QoS metric that is being monitored (collected). The MonitoredDataItem element type is a WS-Policy4MASC supporting construct and it will be described later in this document. The second, optional, child element of the MonitoredDataCollection element type is <*ConfigurationData*>. This element is intended for specifying whatever configuration data the monitoring party needs. Its contents depends upon the monitored data item, but it can contain an arbitrary number of XML elements. For example, for calculated data items, this element could contain an arithmetic with unit expression (<*arithmeticWithUnitExpression*>) describing how to calculate this value. (If during run-time, the specified contents cannot be understood by the MASC middleware, a run-time exception will be thrown.) In principle, this configuration data could include the name of the implementation module to be used for measurement or calculation. However, this brings implementation details into WS-Policy4MASC files, which should be avoided. Therefore, it is better to keep internal tables specifying which ontological definitions (of QoS metrics or context properties) are monitored by which internal modules. The current architecture of the MASC middleware contains 1 generic module for calculations and 2 modules for measurements (one for capturing current time and one for manipulating counters). This covers the most frequently used QoS metrics, such as response time, throughput, and availability. Modules for additional QoS metrics and context properties (e.g., determining geographic coordinates using GPS hardware) are outside the scope of the current version of the MASC middleware. The optional Boolean attribute "PassedInSOAPHeaders" denotes whether the exchange of monitoring data is performed through piggybacking in SOAP headers. Since this is the default way to exchange monitoring data between management parties using the MASC middleware, the default value of the "PassedInSOAPHeaders" attribute is "true".

The definition of this element type (along with the elements <*MonitoredDataCollection*> and <*MonitoredDataCollectionRef*>) is in the file ws-policy4masc-ap.xsd , which is listed in **Appendix 6**.

Figure 11 shows (in the Altova XmlSpy notation) the structure of the **MonitoredDataTransfer element type**. This element type extends MASCConstruct and adds 1 required child element and 2 optional attributes. The required child element (which can be repeated 1 to many times) is a MASCRef reference to a <*MonitoredDataItem*> element that describes the message part or QoS metric that is being transferred between management parties. The optional attribute "TransferPartner" of the XML Schema data type "anyURI" contains the name of the management party to which a monitored data item is sent. Its default value is "MASC_WSORCHESTRATOR", which denotes the Web services orchestrator. Note that the monitored data item is always sent from the party that performs its collection. The optional string attribute "TransferType" captures the type of transfer that is being performed, other than piggybacking in SOAP headers. Its default value is "push", which means that the management party collecting this data item pushes its value to the transfer partner (specified in the "TransferPartner"). Another possible value is "pull", which means that the transfer party queries the management party collecting this data item. Other possible values might be defined in the future. Note that if transfer of monitored data items is performed only through piggybacking in SOAP headers (which is MASC default), then there is no need to specify an instance of the <*MonitoredDataTransfer*> element – the information in the corresponding <*MonitoredDataCollection*> element is enough.

The definition of this element type (along with the elements <*MonitoredDataTransfer*> and <*MonitoredDataTransferRef*>) is in the file ws-policy4masc-ap.xsd , which is listed in **Appendix 6**.

An example of a goal policy assertion is given in **Figure 12**. It is a part of the bigger case study given in **Appendices 1, 2, and 3**. In particular, this XML snippet is from **Appendix 2**. The example in Figure 12 first defines 3 monitored data collection actions – measuring the start time of invocation response time, measuring the stop time of invocation response time, and calculating the response time as the difference

between the latter and the former. The <*ConfigurationData*> child element of the third <*MonitoredDataCollection*> element specifies which measured QoS metrics to use in the calculation of response time. Next, the example in Figure 12 contains definitions of 2 action groups – the first action group contains the first monitored data collection action, while the second action groups contains the second and third monitored data collection action. Finally, the example in Figure 12 contains two action policy assertions. The first action policy assertion (with MASCID "MonitorResponseTimeStart") states that the responsible party is the requester. Further, it refers to the <*When*> construct "Executing-RequestToBeSent" and to the first of the 2 previously defined action groups. The second action policy assertion (with MASCID "MonitorResponseTimeStop") states that the responsible party is the requester. Further, it refers to the <*When*> construct "Executing-ReplyReceived" and the second of the 2 previously defined action groups.

Relationship to the determined requirements for a policy language: These solutions are towards satisfying the requirement 7 "Support for QoS Monitoring".

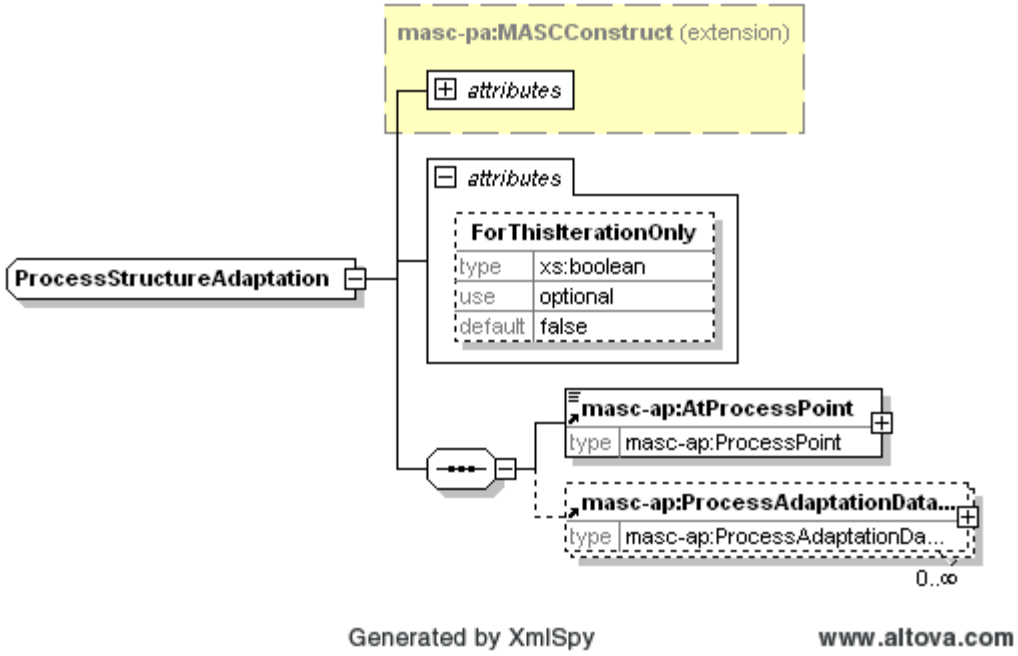

Figure 13. The structure of the ProcessStructureAdaptation type

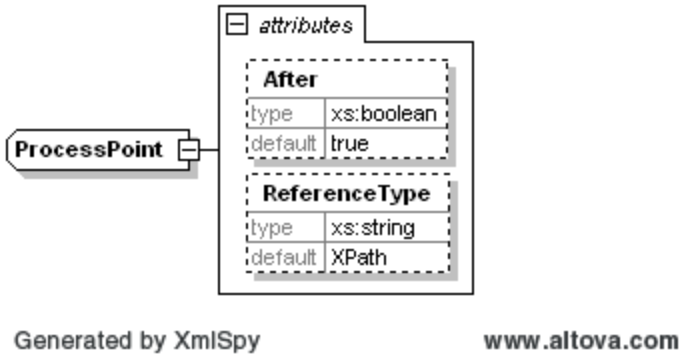

Figure 14. The structure of the ProcessPoint type

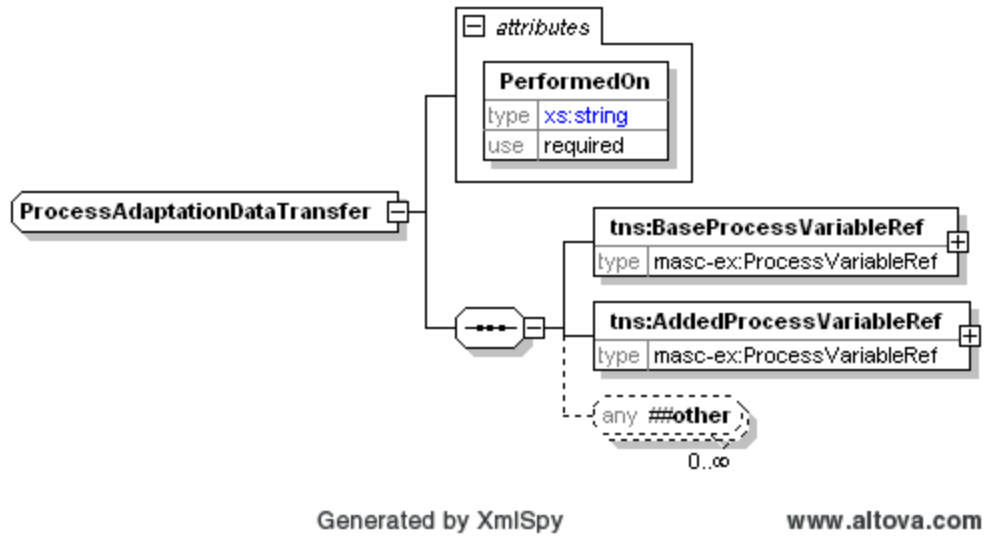

Figure 15. The structure of the ProcessAdaptationDataTransfer type

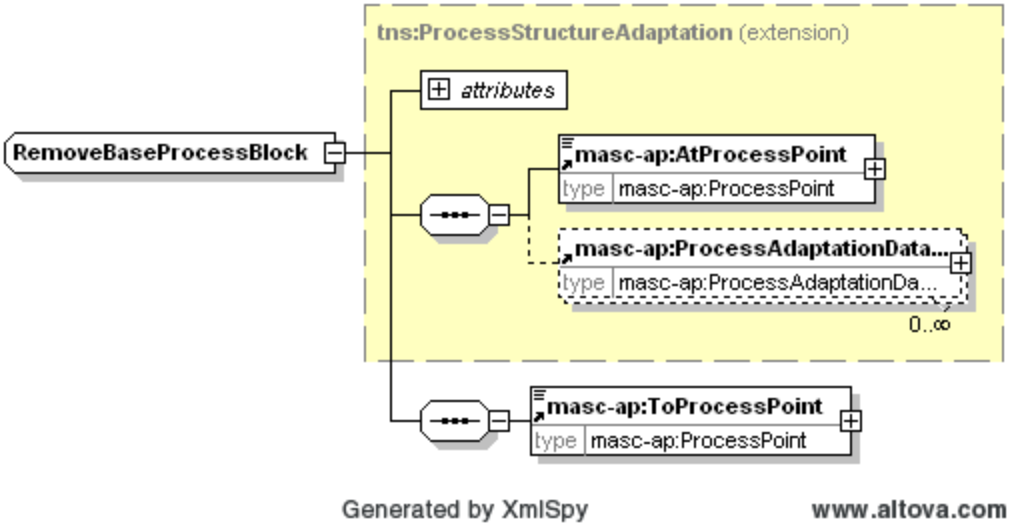

Figure 16. The structure of the RemoveBaseProcessBlock type

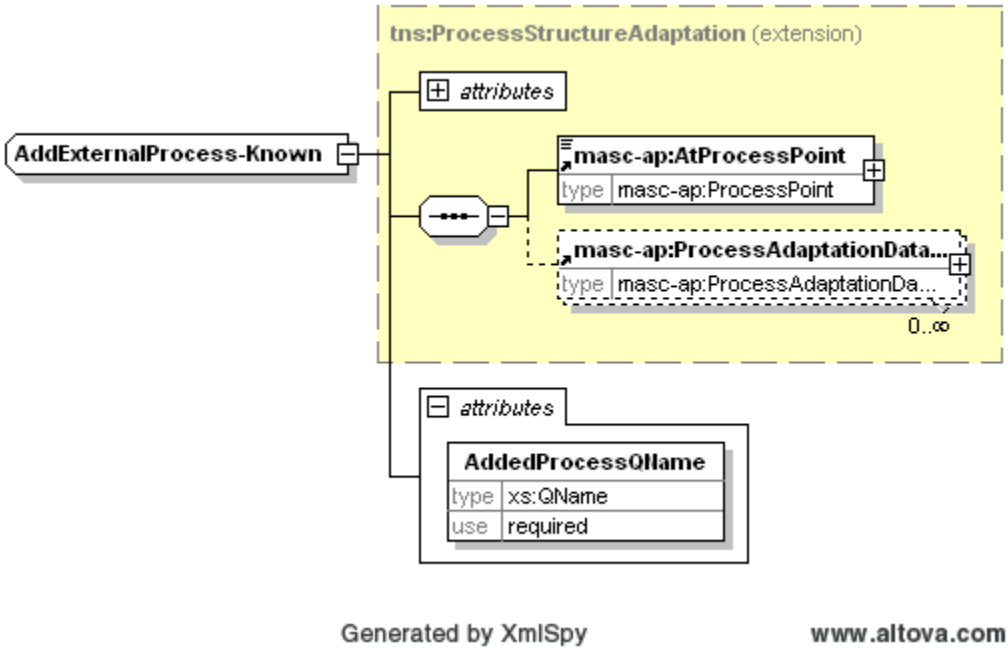

Figure 17. The structure of the AddExternalProcess-Known type

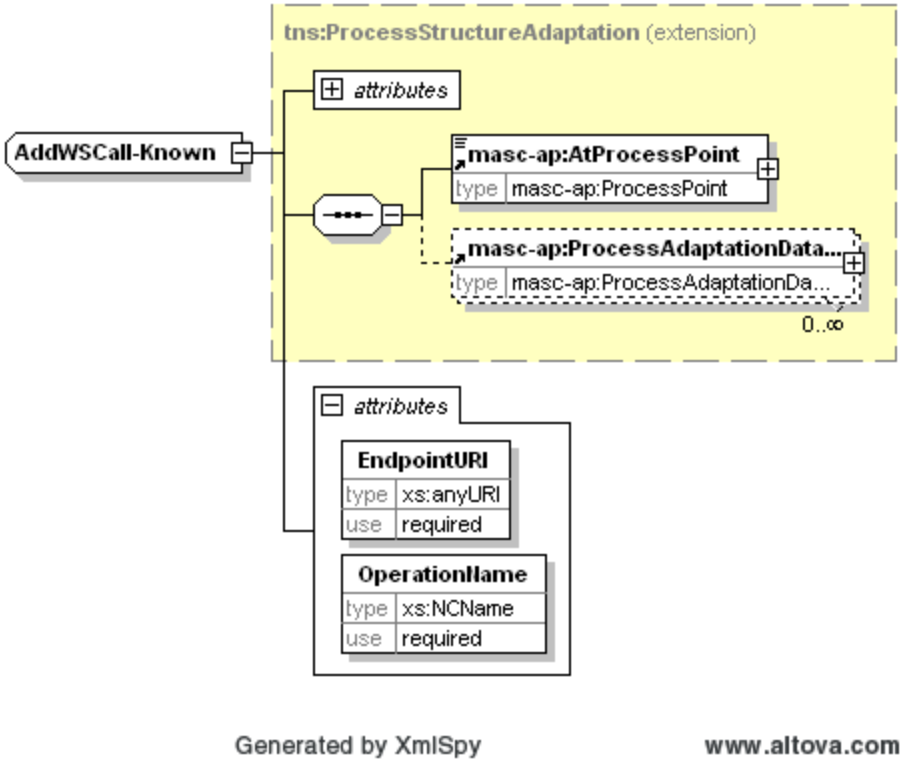

Figure 18. The structure of the AddWSCall-Known type

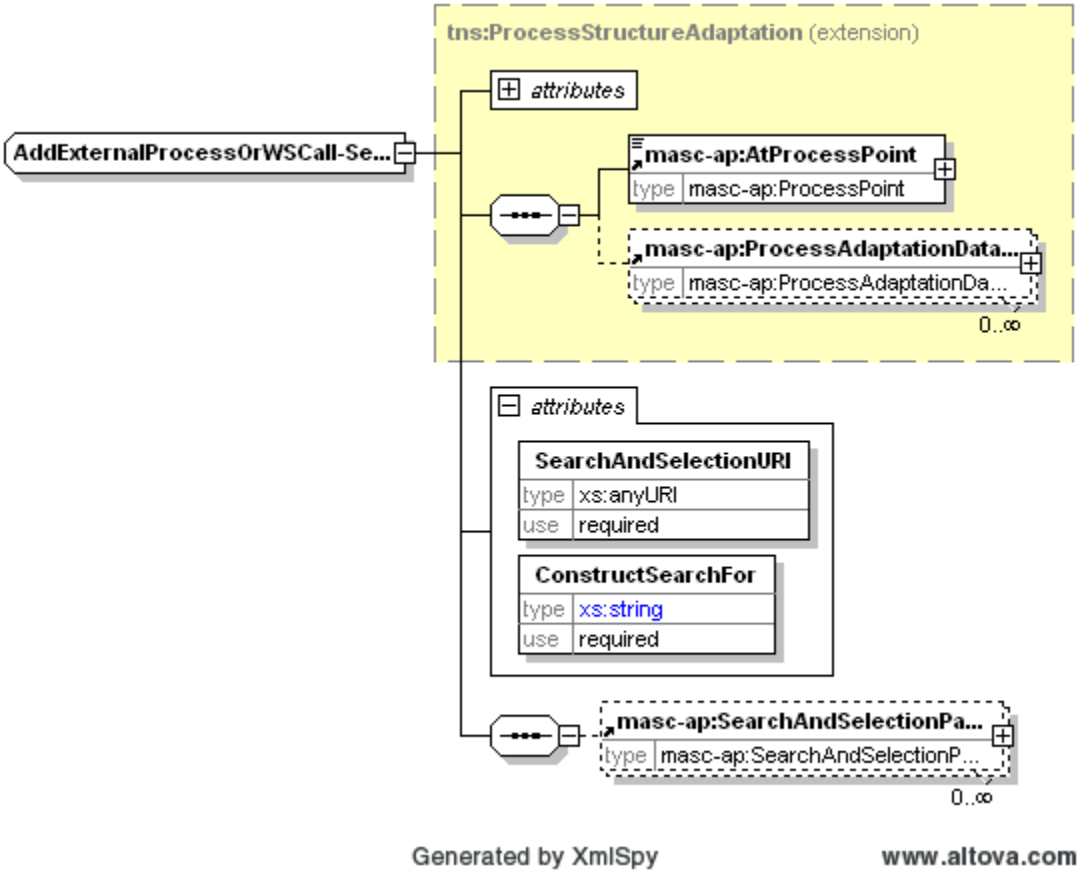

Figure 19. The structure of the AddExternalProcessOrWSCall-Searched type

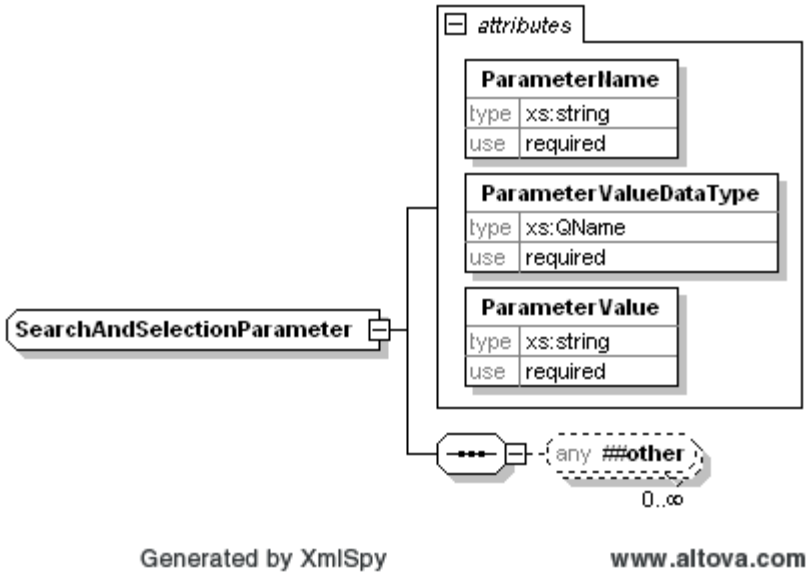

Figure 20. The structure of the SearchAndSelectionParameter type

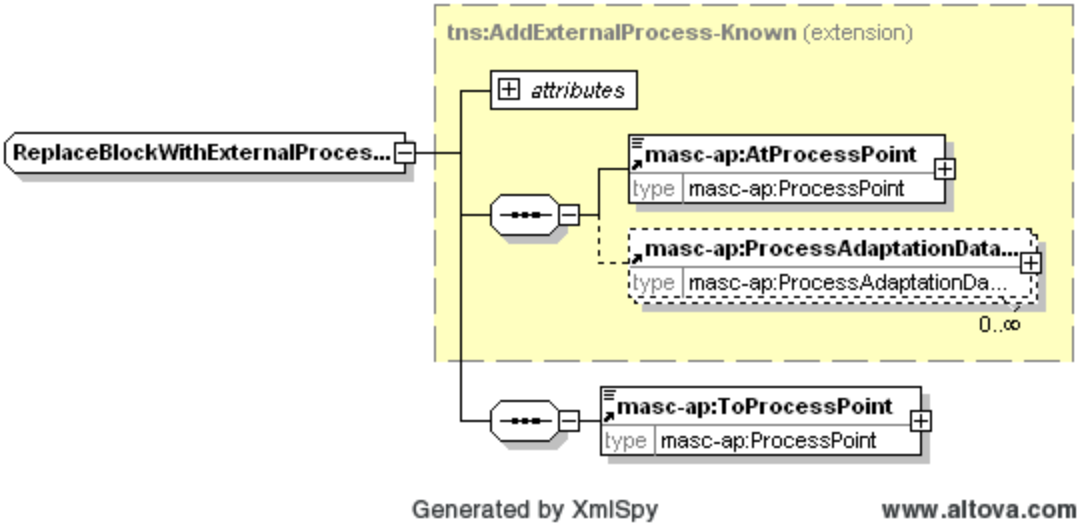

Figure 21. The structure of the ReplaceBlockWithExternalProcess-Known type

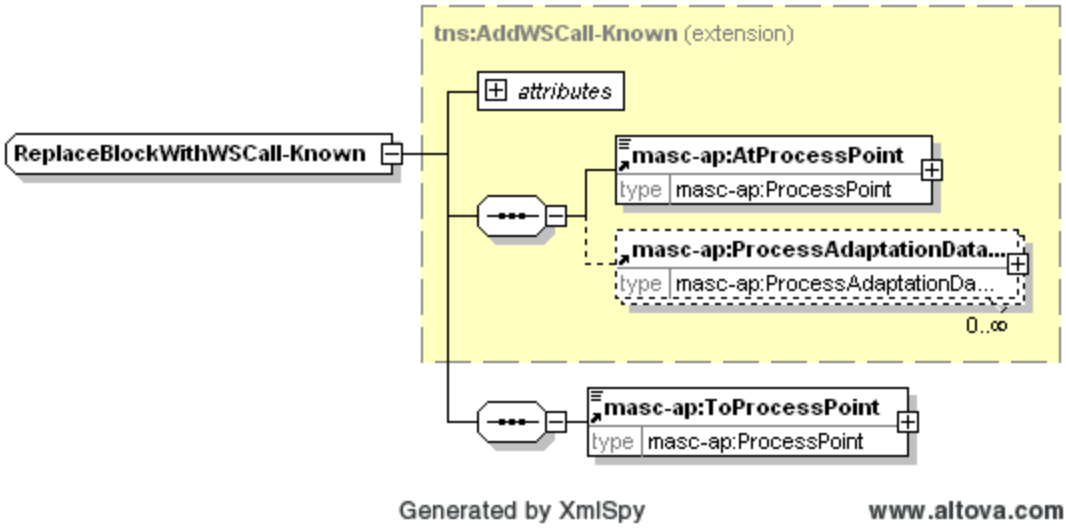

Figure 22. The structure of the ReplaceBlockWithWSCall-Known type

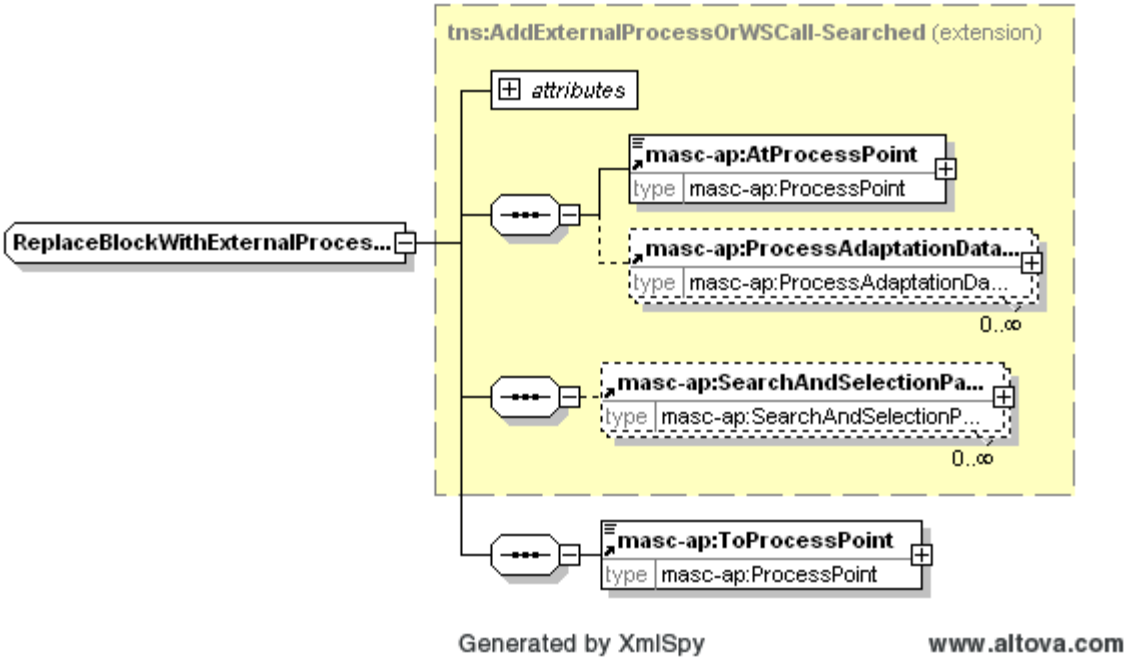

Figure 23. The structure of the ReplaceBlockWithExternalProcessOrWSCall-Searched type

```
<masc-ap:ReplaceBlockWithWSCall-Known MASCID="ProviderWSReplacement"
EndpointURI="http://www.fasterWeatherReport.com/Canada" OperationName="WeatherTemperature">
  <masc-ap:AtProcessPoint After="true"
ReferenceType="MASC_PROCESSRELATIVE">MASC_PROCESSBEGINNING</masc-ap:AtProcessPoint>
  <masc-ap:ToProcessPoint After="false"
ReferenceType="MASC_PROCESSRELATIVE">MASC_PROCESSENDING</masc-ap:ToProcessPoint>
 </masc-ap:ReplaceBlockWithWSCall-Known>
```

Figure 24. Example of a process structure adaptation replacing a block of the base process with a call to a known Web service

ProcessStructureAdaptation, its sub-types, and related constructs:

The abstract ProcessStructureAdaptation element type, its sub-types (extensions), types that are referenced by ProcessStructureAdaptation or its subtypes, and elements corresponding to any of the previously listed types are used to describe adaptation of the process (workflow, Web service composition) structure. There are 3 broad categories of such adaptation: removal (deletion) of a part of the process, addition into the process, and replacement of a part of the process. In each of these 3 categories several versions are possible, depending on what is being removed/added/replaced. Note that ProcessStructureAdaptation is intended as an abstract super-type for element types that describe particular versions of process structure adaptation. We will now describe ProcessStructureAdaptation, its sub-types, and related constructs. Hereafter, we will use the term “base process” to denote the process that is being modified, the term “added process” to denote the addition or replacement, and the term “process variation” to denote removal, addition, or replacement (thus, every “added process” is also a “process variation”).

Figure 13 shows (in the Altova XmlSpy notation) the structure of the **ProcessStructureAdaptation element type**. This element type extends MASCConstruct and adds 1 required child element, 1 optional child element, and 1 optional attribute. The first, required, child element is <*AtProcessPoint*>, which describes at which point in the process the described structure adaptation should be applied. It is an instance of the ProcessPoint element type described below. The second, optional, child element describes passing of values between the base process and the added process (or between two points in the base process that become one point after a removal of some process block). It is an instance of the ProcessAdaptationDataTransfer element type described below. The optional Boolean attribute “ForThisIterationOnly” is relevant for structural adaptations performed within a loop (while for structural adaptations outside loops it makes no difference). If it is “false” (which is default), the structural adaptation is performed for all subsequent iterations of the containing loop. On the other hand, if it is “true”, then the

structural adaptation is performed only for 1 next iteration of the containing loop.

The definition of this element type (along with the element <*ProcessStructureAdaptationRef*>) is in the file ws-policy4masc-ap.xsd , which is listed in **Appendix 6**.

Figure 14 shows (in the Altova XmlSpy notation) the structure of the **ProcessPoint element type**. Note that this element type does not extend MASCConstruct or any of its sub-types, so its instances must not be specified as WS-Policy policy assertions – they may be specified only within other WS-Policy4MASC elements. An instance of this element type is used to denote point within a process (workflow, Web service composition). ProcessPoint defines two optional attributes, while its value (treated as a string) contains the actual description of the specified point. The Boolean attribute "After" denotes whether the structural adaptation should be applied after or before the specified point. Its default is "true". The meaning of this attribute depends on whether the specified point is a point of process addition, beginning of a block to be removed or replaced, or ending of a block to be removed or replaced. For example, if the value is "true" and the specified point is used for process addition, this means that addition is performed after the specified point. The string attribute "ReferenceType" specifies the format in which the point is described. Its default is XPath, but other formats are possible. Our examples often use the simple format "MASC_PROCESSRELATIVE" in which the described point is either "MASC_PROCESSBEGINNING" or "MASC_PROCESSENDING". This models two common special cases: process beginning (before the first activity) and process ending (after the last activity). Some examples of use of elements of the ProcessPoint type are given in **Figure 24** and explained below.

The definition of this element type (along with the instance elements <*AtProcessPoint*> and <*ToProcessPoint*>) is in the file ws-policy4masc-ap.xsd , which is listed in **Appendix 6**.

Figure 15 shows (in the Altova XmlSpy notation) the structure of the **ProcessAdaptationDataTransfer element type**. Note that this element type does not extend MASCConstruct or any of its sub-types, so its instances must not be specified as WS-Policy policy assertions – they may be specified only within other WS-Policy4MASC elements. An instance of this element type is used to specify how to transfer values between the base process and added process (or between two parts of the base process that are joined after a removal of a block from the base process). ProcessAdaptationDataTransfer defines 2 required child elements, 0 to many optional elements, and 1 required attribute. The required child elements <*BaseProcessVariableRef*> and <Added*ProcessVariableRef*> are both references to process variables (the type ProcessVariableRef will be explained later). The former is a reference to a variable in the base process, while the latter is a reference to a variable in the added process (or in the part of the base process that comes after a removed block). A WS-Policy4MASC parser should perform a semantic check that the data types of the 2 specified variables are the same (alternatively, it could be checked that they are compatible, but this is more complex). In this version of WS-Policy4MASC, only direct transfers between 2 variables, without any data transformations, is supported. Since data transformation might be needed in some cases, we have allowed 0 to many optional elements of any type, to serve as language extensibility points. (Due to the complexity of the required support, MASC middleware prototype does not yet support data transformations and will probably not support them in the near future.) The required attribute "PerformedOn" specifies when to perform this data transfer. It is a restriction of the XML Schema string data and can have one of the following (enumeration) values: "CallToProcessVariationOnly", "ReturnFromProcessVariationOnly", "BothCallToAndReturnFromProcessVariation", or "Always-BoundVariables". WS-Policy4MASC supports two different data transfer patterns: copying of values between variables and variable binding. Copying of values is specified using one of the first 3 values for the "PerformedOn" attribute. It has to be defined separately for the data transfer from the base process to the process variation (at the variation process initialization) and for the data transfer from the process variation to the base process (at the process variation finalization). On the other hand, variable binding is specified using the last value for the "PerformedOn" attribute. It has to be defined only at the data transfer from the base process to the process variation and it remains valid afterwards (even after the data transfer from the process variation to the base process).

The definition of this element type (along with the instance element <*ProcessAdaptationDataTransfer*>) is in the file ws-policy4masc-ap.xsd , which is listed in **Appendix 6**.

Figure 16 shows (in the Altova XmlSpy notation) the structure of the **RemoveBaseProcessBlock element type**. This element type extends ProcessStructureAdaptation and adds 1 required child element <*ToProcessPoint*> of the ProcessPoint element type. It describes the ending point of the base process block to be removed (deleted), while the inherited child element <*AtProcessPoint*> describes the beginning point of the base process block to be removed. Note that we have considered adding another element type RemoveBaseProcessActivity (also as an extension of ProcessStructureAdaptation) to model the case when only one process activity is removed. However, this is semantically only a special case of

RemoveBaseProcessBlock and can be modeled with it easily (although somewhat verbosely). Therefore, the current version of WS-Policy4MASC contains only RemoveBaseProcessBlock.

The definition of this element type (along with the elements <*RemoveBaseProcessBlock*> and <*RemoveBaseProcessBlockRef*>) is in the file ws-policy4masc-ap.xsd , which is listed in **Appendix 6**.

Figure 17 shows (in the Altova XmlSpy notation) the structure of the **AddExternalProcess-Known element type**. This element type is used for the type of process addition when another, already known, process is added into the base process. This element type extends ProcessStructureAdaptation and adds 1 required attribute "AddedProcessQName" which contains a namespace qualified name of the added process. This is a reference to the description (e.g., in WS-BPEL or XAML) of the added process.

The definition of this element type (along with the elements <*AddExternalProcess-Known*> and <*AddExternalProcess-KnownRef*>) is in the file ws-policy4masc-ap.xsd , which is listed in **Appendix 6**.

Figure 18 shows (in the Altova XmlSpy notation) the structure of the **AddWSCall-Known element type**. This element type is used for an important special type of process addition when only a call to a known Web service is added into the base process. The advantage of this special case is that MASC middleware could automatically generate the client stub for this Web service call, so client code in C# need not exist beforehand. (This feature is in the MASC architecture, but not yet in the MASC prototype.) The AddWSCall-Known element type extends ProcessStructureAdaptation and adds 2 required attributes. The attribute "EndpointURI" (of the anyURI data type) captures the URI of the Web service endpoint to be called, while the attribute "OperationName" (of the XML Schema data type NCName, represented names without namespaces) captures the name of the called operation. Note that values to operation parameters are passed through the inherited element <*ProcessAdaptationDataTransfer*>.

The definition of this element type (along with the elements <*AddWSCall-Known*> and <*AddWSCall-KnownRef*>) is in the file ws-policy4masc-ap.xsd , which is listed in **Appendix 6**.

Figure 19 shows (in the Altova XmlSpy notation) the structure of the **AddExternalProcessOrWSCall-Searched element type**. This element type is used for a special type of process addition when the added process (or, in a special sub-case, added call to a Web service) is not known beforehand and has to be searched in some repository and selected using some criteria. The advantage of this special case is that MASC middleware could automatically engage in such search and selection. (This feature is in the MASC architecture, but not yet in the MASC prototype.) The AddExternalProcessOrWSCall-Searched element type extends ProcessStructureAdaptation and adds 0 to many optional elements and 2 required attributes. The optional elements are of type SearchAndSelectionParameter (described below) and they describe parameters for the search and selection process. The required attribute "SearchAndSelectionURI" (of the anyURI data type) captures the URI of the repository to be searched. The required attribute "ConstructSearchedFor" distinguishes whether the searched construct is a process or a Web service operation, so its type is restricted string (enumeration) with 2 possible values "Process" and "WebServiceOperation".

The definition of this element type (along with the elements <*AddExternalProcessOrWSCall-Searched*> and <*AddExternalProcessOrWSCall-SearchedRef*>) is in the file ws-policy4masc-ap.xsd , which is listed in **Appendix 6**.

Figure 20 shows (in the Altova XmlSpy notation) the structure of the **SearchAndSelectionParameter element type**. Note that this element type does not extend MASCConstruct or any of its sub-types, so its instances must not be specified as WS-Policy policy assertions – they may be specified only within other WS-Policy4MASC elements. An instance of this element type specifies a parameter used in the search and selection process for the previously described process structure adaptation. This element type defines 0 to many optional elements and 3 required attributes. The optional elements can be any XML elements and are intended to be extensibility points for future complex assignment of parameter values. Currently, only simple assignment of parameter values is supported and it is performed through the 3 required attributes "ParameterName" (of string data type), "ParameterValueDataType" (of QName data type), and "ParameterValue" (of string data type).

The definition of this element type (along with the instance element <*SearchAndSelectionParameter*>) is in the file ws-policy4masc-ap.xsd , which is listed in **Appendix 6**.

Figures 21, 22, and 23 show (in the Altova XmlSpy notation) the structure of the **ReplaceBlockWithExternalProcess-Known, ReplaceBlockWithWSCall-Known, and ReplaceBlockWithExternalProcessOrWSCall-Searched element types**, respectively. Replacement of a process block with another process (or call to a Web service) can be modeled as a combination of a removal and an addition. However, , we decided to model it explicitly because it is a common case and the meaning of replacement is hidden in such modeling. In our implementation, ReplaceBlockWithExternalProcess-Known, ReplaceBlockWithWSCall-Known, and ReplaceBlockWithExternalProcessOrWSCall-Searched

element types extend AddExternalProcess-Known, AddWSCall-Known, and AddExternalProcessOrWSCall-Searched element types, respectively. They all add only 1 required child element <*ToProcessPoint*>, which is of the ProcessPoint element type. It describes the ending point of the base process block to be replaced, while the inherited child element <*AtProcessPoint*> describes the beginning point of the base process block to be replaced.

The definitions of these element types (along with the elements <*ReplaceBlockWithExternalProcess-Known*>, <*ReplaceBlockWithWSCall-Known*>, <*ReplaceBlockWithExternalProcessOrWSCall-Searched*>, <*ReplaceBlockWithExternalProcess-KnownRef*>, <*ReplaceBlockWithWSCall-KnownRef*>, and <*ReplaceBlockWithExternalProcessOrWSCall-SearchedRef*>) are in the file ws-policy4masc-ap.xsd , which is listed in **Appendix 6**.

An example of a goal policy assertion is given in **Figure 24**. It is a part of the bigger case study given in **Appendices 1, 2, and 3**. In particular, this XML snippet is from **Appendix 3**. The example in **Figure 24** defines one action of the <*ReplaceBlockWithWSCall-Known*> variety. It MASCID is "ProviderWSReplacement", the endpoint of the called Web service is "http://www.fasterWeatherReport.com/Canada", and operation name is "WeatherTemperature". The value of the optional attribute "ForThisIterationOnly" is not specified, so the default value "false" is used. The replaced block begins immediately after the process beginning, which is specified in the <*AtProcessPoint*> element, its attributes, and its value (specified in our simple format for description of process beginning and ending). The replaced block ends immediately before the process ending, which is specified in the <*AtProcessPoint*> element, its attributes, and its value (specified in our simple format for description of process beginning and ending). This means that the call to this Web service operation replaces the whole previously defined process.

Relationship to the determined requirements for a policy language: These solutions are towards satisfying the requirements 13 "Support for Dynamic Adaptation", 15 "Specification of Static Customization of Web Service Composition", and 16 "Specification of Dynamic Addition/Deletion/Replacement in Web Service Compositions".

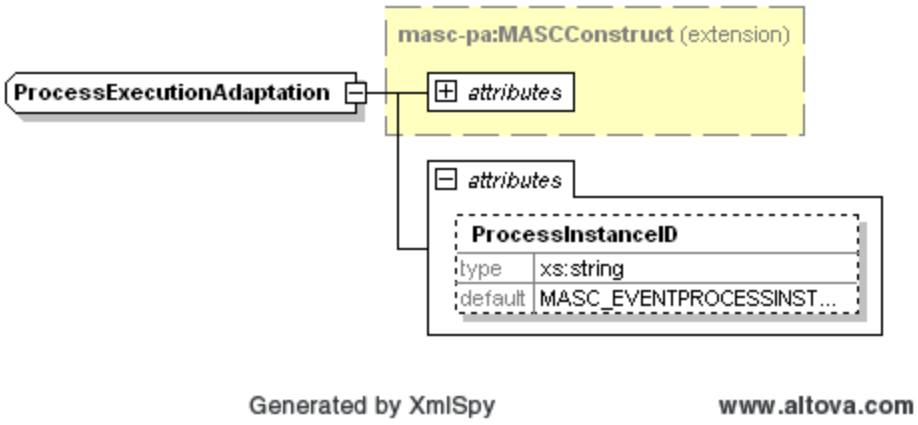

Figure 25. The structure of the ProcessExecutionAdaptation type

Figure 26. The structure of the TerminateProcess type

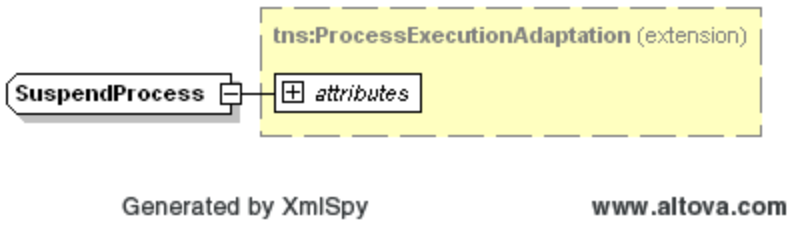

Figure 27. The structure of the SuspendProcess type

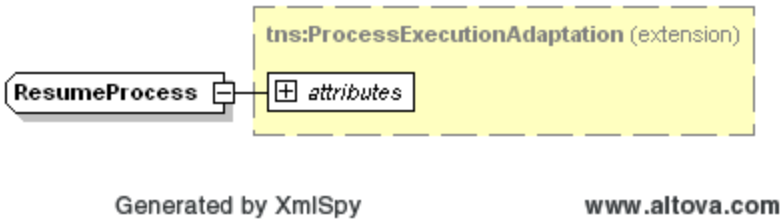

Figure 28. The structure of the ResumeProcess type

ProcessExecutionAdaptation and its sub-types:

The abstract ProcessExecutionAdaptation element type and its sub-types (extensions), and elements corresponding to these types are used to describe adaptation of the process (workflow, Web service composition) execution at the level of a process instance. There are 3 versions of such adaptation: **termination, suspension, and resumption**. Note that ProcessExecutionAdaptation is intended as an abstract super-type for element types that describe particular versions of process execution adaptation. Process termination is modeled with the TerminateProcess element type, process suspension with the SuspendProcess element type, and process resumption with the ResumeProcess element type.

Figure 25 shows (in the Altova XmlSpy notation) the structure of the abstract **ProcessExecutionAdaptation element type**. It extends MASCConstruct by adding 1 optional attribute (and no child element). The optional attribute "ProcessInstanceID" (of the string data type) is needed because an action policy could be attached to a process type and several instances of this process type could execute at the same time in the managed system. In the majority of cases the process instance that is adapted (particularly: terminated or suspended) is the one for which the triggering event for the containing action policy assertion is raised. (Clarification: The action policy that contains the specified process execution adaptation action must have a <*When*> element, which lists one or more triggering events. During run-time, one of these events is raised and it contains some run-time information, including the ID of the process instance for which it is raised. In the majority of cases, this is the process instance that should be terminated/suspended/resumed.) This situation is captured in the default value of the "ProcessInstanceID" attribute, the constant "MASC_EVENTPROCESSINSTANCE". If a WS-Policy4MASC policy execution adaptation element contains some other, non-default, value for this attribute, MASC middleware should check that such an instance exists (and raise a run-time exception if it does not). In a future version of the WS-Policy4MASC language, new attributes (or elements) might be added to enable specification of information about where to log and/or send a notification about the performed process execution adaptation.

The definition of this element type (along with the element <*ProcessExecutionAdaptationRef*>) is in the file ws-policy4masc-ap.xsd , which is listed in **Appendix 6**.

Figures 26, 27, and 28 show (in the Altova XmlSpy notation) the structure of the **TerminateProcess, SuspendProcess, and ResumeProcess element types**, respectively. In the current version of WS-Policy4MASC, their structure is simple – they extend ProcessExecutionAdaptation without adding any child element or attribute. Information about which process execution action to execute is conveyed through the name of the action to be executed. It might be useful to note that the default value for the "ProcessInstanceID" attribute is usually appropriate for process termination and process suspension, but not always for process resumption. For process instance resumption, MASC middleware should check that this instance was suspended previously (and raise a run-time exception if it was not).

The definition of these element types (along with the elements <*TerminateProcess*>, <*SuspendProcess*>, <*ResumeProcess*>, <*TerminateProcessRef*>, <*SuspendProcessRef*>, and <*ResumeProcessRef*>) is in the file ws-policy4masc-ap.xsd , which is listed in **Appendix 6**.

Relationship to the determined requirements for a policy language: These solutions are towards

satisfying the requirement 13 “Support for Dynamic Adaptation”.

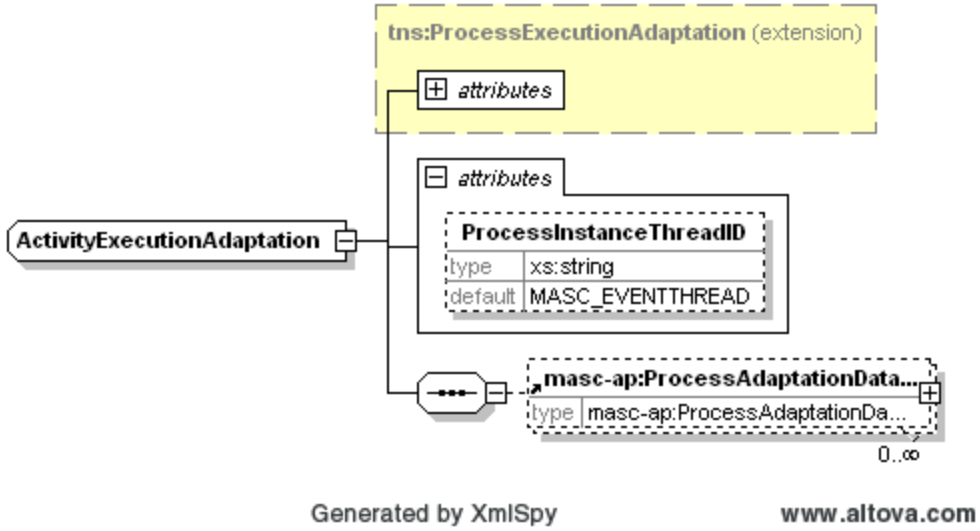

Figure 29. The structure of the ActivityExecutionAdaptation type

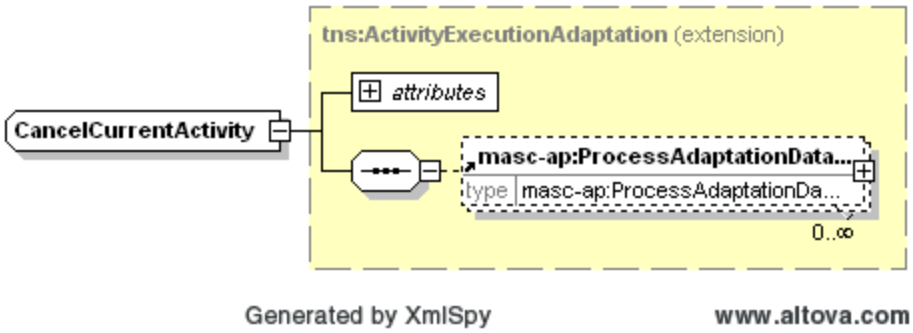

Figure 30. The structure of the CancelCurrentActivity type

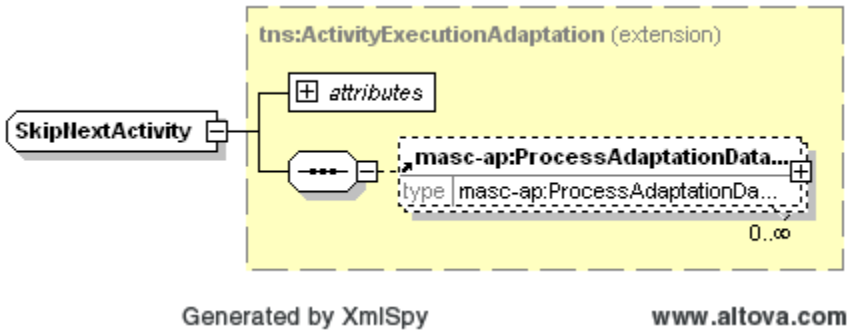

Figure 31. The structure of the SkipNextActivity type

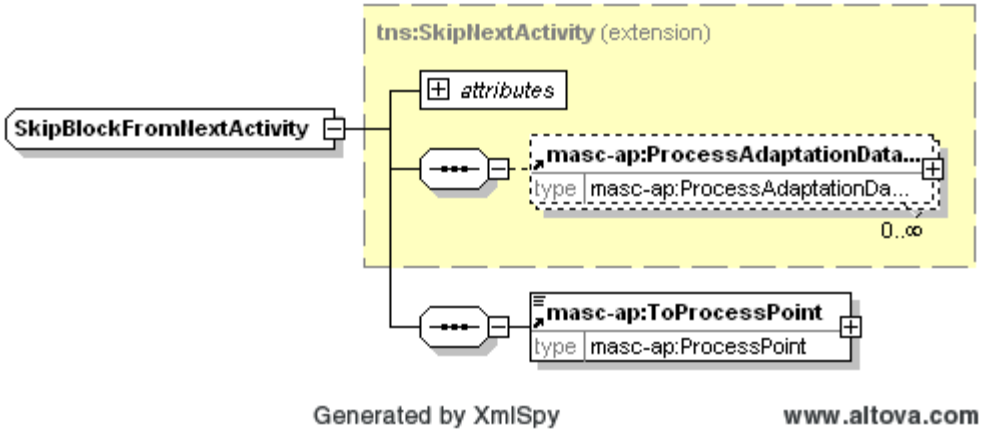

Figure 32. The structure of the SkipBlockFromNextActivity type

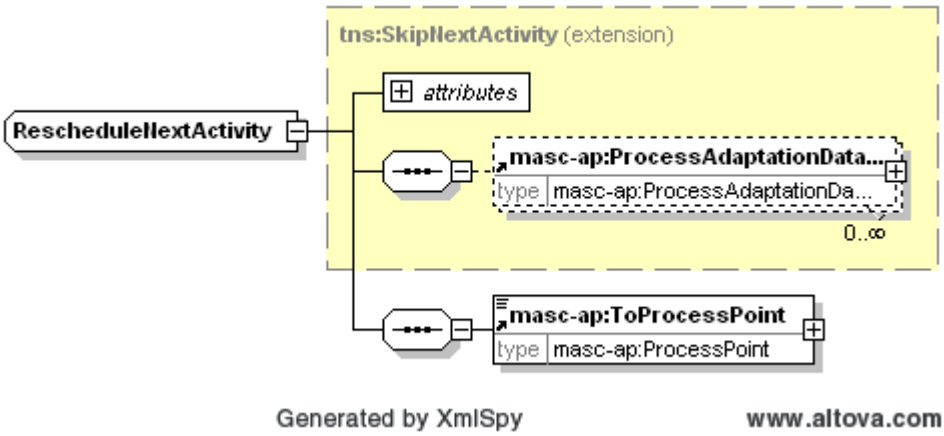

Figure 33. The structure of the RescheduleNextActivity type

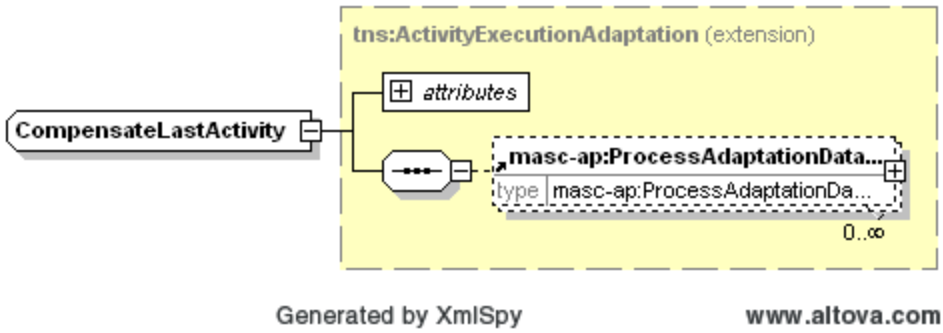

Figure 34. The structure of the CompensateLastActivity type

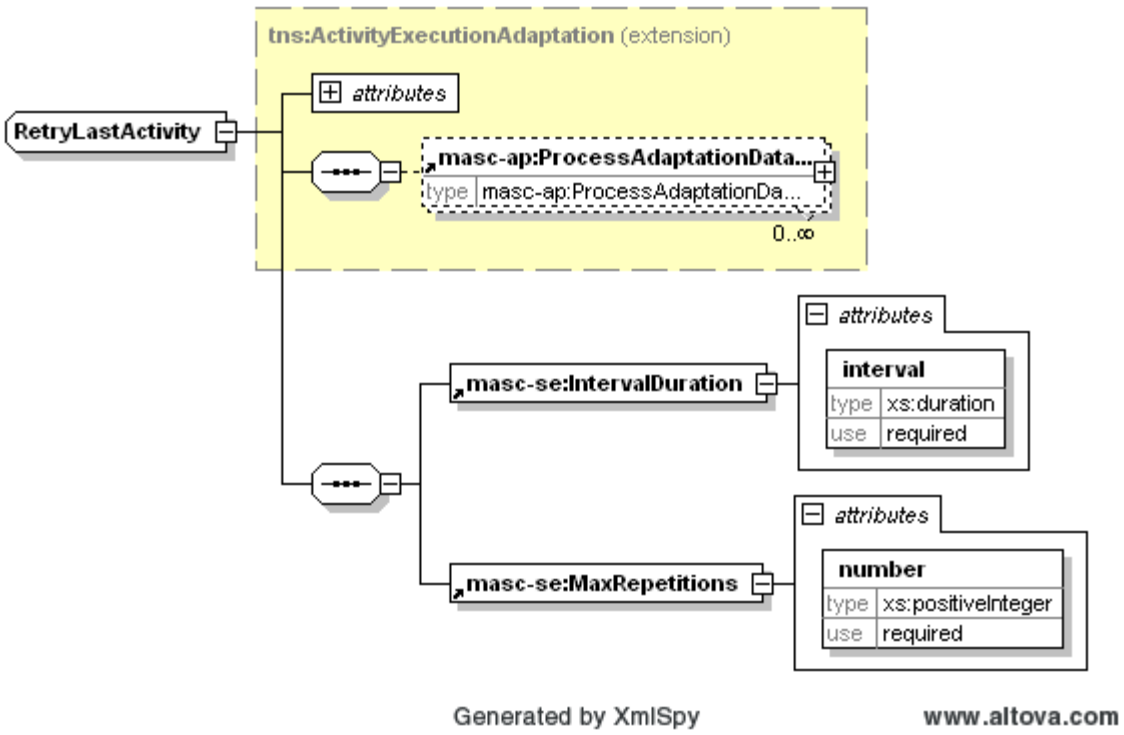

Figure 35. The structure of the RetryLastActivity type

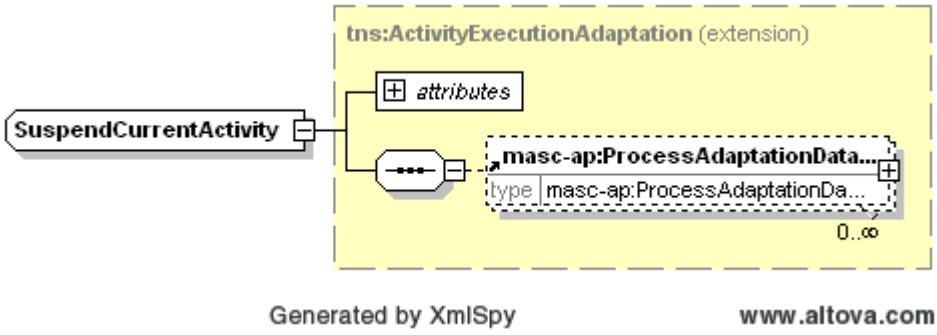

Figure 36. The structure of the SuspendCurrentActivity type

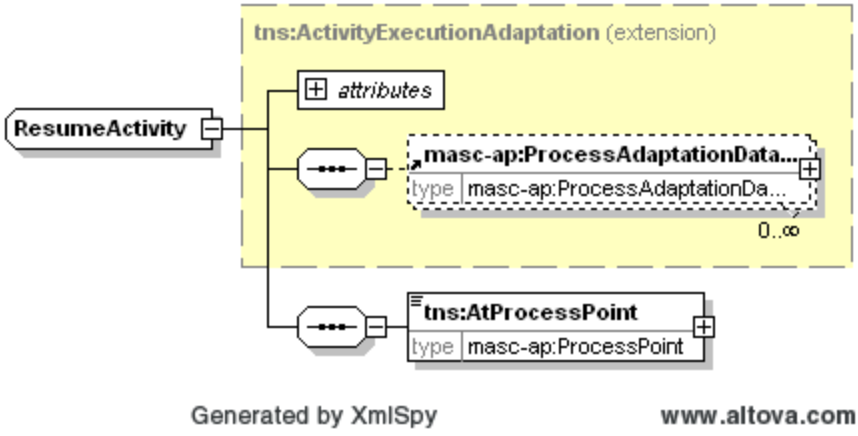

Figure 37. The structure of the ResumeActivity type

ActivityExecutionAdaptation and its sub-types:

The abstract ActivityExecutionAdaptation element type and its sub-types (extensions), and elements corresponding to these types are used to describe adaptation of the process (workflow, Web service

composition) execution at the level of a particular activity within a process instance. There are 8 versions of such adaptation: **canceling the current activity, skipping the next activity, skipping a block starting with the next activity, rescheduling the next activity, compensating the last activity, retrying the last activity, suspend current activity, and resuming (previously suspended) activity**. Note that ActivityExecutionAdaptation is intended as an abstract super-type for element types that describe particular versions of activity execution adaptation.

Note that action policies containing activity execution adaptations are attached to process types or process instances. We have considered attaching them to a particular activity within a process type (or process instance). However, this would require bigger changes in WS-Policy4MASC to separate different actions for different attachment areas. Also, the meaning of attaching some of these actions to activities in different execution stages (e.g., attaching compensation to an activity that was not yet executed or attaching skipping to an already executed activity) is not clear. Therefore, we have assumed that the containing action policies are attached to process types or process instances and defined activity execution adaptation actions to be applied at (or next to) the current execution point of the process instance. (We have also accommodated the case of multiple threads/forks within a process, as will be explained below.) If adaptation should be applied to another activity, then a corresponding process structure adaptation construct with the value "true" for the attribute "ForThisIterationOnly" must be used.

Figure 29 shows (in the Altova XmlSpy notation) the structure of the abstract **ActionExecutionAdaptation element type**. It extends ProcessExecutionAdaptation by adding 0 to many child elements and 1 optional attribute. The optional <*ProcessAdaptationDataTransfer*> child elements are used to specify how to transfer values between the previous point in the process execution and the new point in the process execution. For example, if an activity is skipped, it might be needed to relate values of some process variables after this activity to values of these and/or other process variables before this activity. The optional attribute "ProcessInstanceThreadID" (of the string data type) is needed because a process instance might have several concurrent threads (forks) between which we have to distinguish. In the majority of cases the activity that is adapted is the one for which the triggering event for the containing action policy assertion is raised. This situation is captured in the default value of the "ProcessInstanceThreadID" attribute, the constant "MASC_EVENTTHREAD". If a WS-Policy4MASC policy execution adaptation element contains some other, non-default, value for this attribute, MASC middleware should check that such a thread exists (and raise a run-time exception if it does not). Note that the attribute "ProcessInstanceThreadID" complements the inherited attribute "ProcessInstanceID".

The definition of this element type (along with the element <*ActivityExecutionAdaptationRef*>) is in the file ws-policy4masc-ap.xsd , which is listed in **Appendix 6**.

Figures 30 through 37 show (in the Altova XmlSpy notation) the structure of sub-types of the ActionExecutionAdaptation element type. These are (in the same order as in the figures): C**ancelCurrentActivity, SkipNextAtivity, SkipBlockFromNextActivity, RescheduleNextActivity, CompensateLastActivity, RetryLastActivity, SuspendCurrentActivity, and ResumeActivity element types**. They are all relatively simple extensions of ActionExecutionAdaptation and have meaningful names. We think that there is no need to go into a detailed explanation of their every child element and attribute. However, a we have to make a few notes. First, it is important to recognize that the name of each of these actions (except <*ResumeActivity*>) contains a word "current", "next", or "last". They refer, respectively, to the activity (in the specified process instance and thread) that is still executing , that will execute when the current one finishes, and that has last completed execution regularly or with some error. Each of the listed actions (except <*SkipBlockFromNextActivity*> and <*ResumeActivity*>) applies to 1 of these activities. Currently executing activity can be suspended or canceled, next activity can be skipped or rescheduled, while erroneous last activity can be compensated or retried. Second, skipping is different from deletion, because when the process block is in a loop, skipping affects only the current iteration and subsequent iterations are not affected, while deletion affects both the current and the subsequent iterations. Third, cancellation of the current activity can be through either invocation of some special operation (we assume it is known to the MASC middleware) or simple skipping of the rest of this activity, depending on the nature of the activity. It seems that an activity that provides a canceling operation should not be simply skipped. Therefore, the current version of WS-Policy4MASC does not support a separate "SkipCurrentActivity" action. Instead, <*CancelCurrentActivity*> should be used, while the MASC middleware would make a decision what this cancellation means. fourth (and somewhat related to the third), compensation is performed by invocation of some special operation (we assume it is known to the MASC middleware). If there is no compensation and/or cancellation activity known to the MASC middleware, then authors of WS-Policy4MASC policies have to use process structure adaptation (e.g., <*AddWSCall-Known*>) to insert desired compensation and/or

cancellation activities (operations, processes) known to them. Fifth, the difference between suspension/resumption of an activity and a whole process is that if a process has more than one concurrent thread (fork), suspending/resuming an activity suspends/resumes only its thread and not the other threads.

The definitions of these element types (along with the corresponding instance elements and MASCRef instances) are in the file ws-policy4masc-ap.xsd , which is listed in **Appendix 6**.

Relationship to the determined requirements for a policy language: These solutions are towards satisfying the requirements 13 “Support for Dynamic Adaptation”, 16 “Specification of Dynamic Addition/Deletion/Replacement in Web Service Compositions”, 18 “Specification of Invocation Retries”, and (to some extent) 17 “Specification of Dynamic Changes of Selection/Iteration Conditions in Web Service Compositions”.

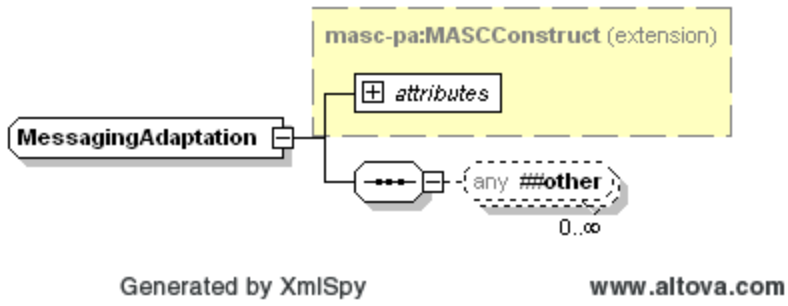

Figure 38. The structure of the MessagingAdaptation type

MessagingAdaptation:

Definition of the abstract MessagingAdaptation element type, its subtypes (e.g., merging, splitting, transforming, enriching and buffering of messages), and corresponding elements is left for a future WS-Policy4MASC version. This has high priority.

The definition of this element type (along with the corresponding instance elements and MASCRef instances) will be in the file ws-policy4masc-ap.xsd , which is listed in **Appendix 6**.

Relationship to the determined requirements for a policy language: These solutions are towards satisfying the requirement 13 “Support for Dynamic Adaptation”.

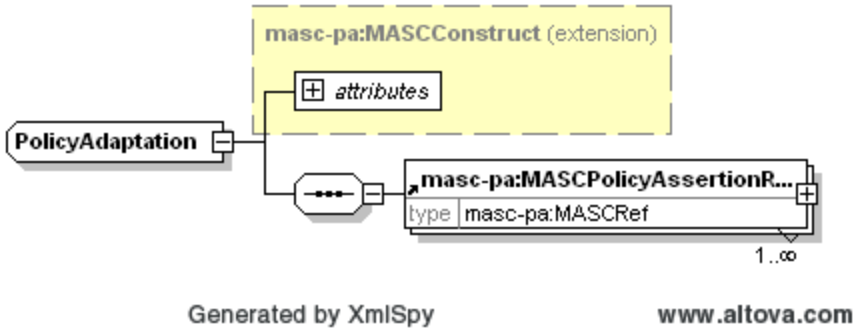

Figure 39. The structure of the PolicyAdaptation type

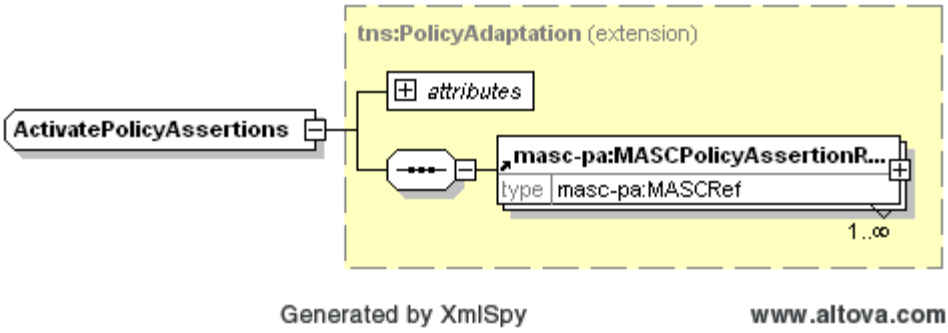

Figure 40. The structure of the ActivatePolicyAssertions type

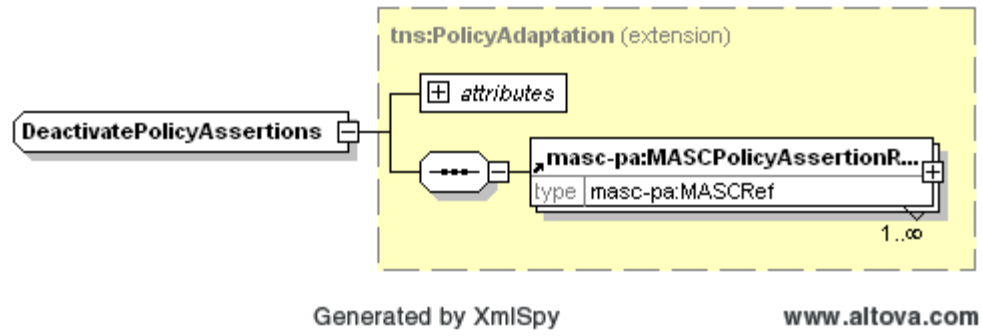

Figure 41. The structure of the DeactivatePolicyAssertions type

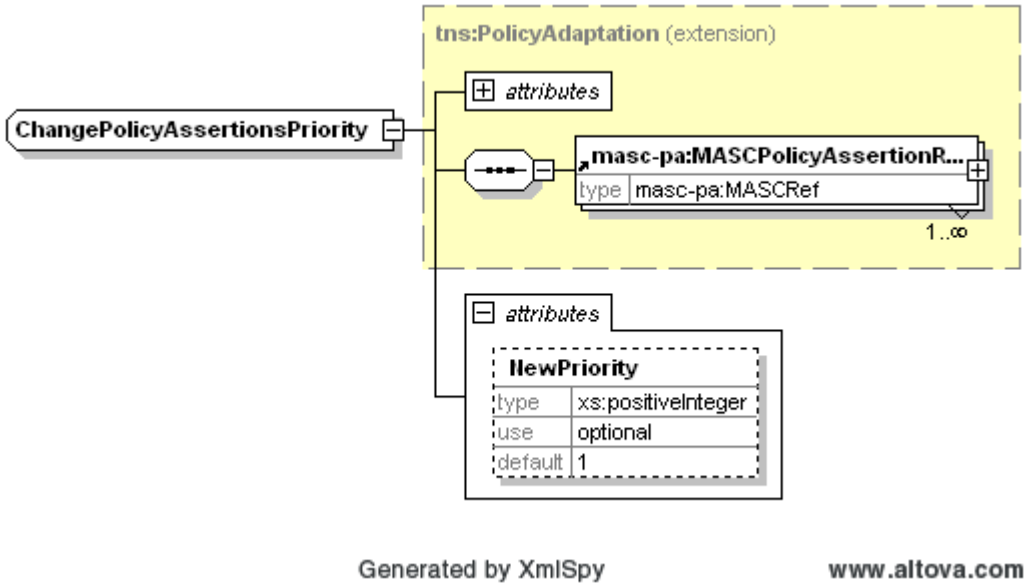

Figure 42. The structure of the ChangePolicyAssertionsPriority type

PolicyAdaptation and its sub-types:

Adaptation of policies by manipulation of policy assertions is a powerful way of adaptation. However, it could also have dangerous consequences (e.g., leaving the management system in an inconsistent state). The current version of WS-Policy4MASC supports only activation of policy assertions, deactivation of policy assertions, and change of priority of policy assertions. We have also considered language-level support for policy addition and policy deletion, but concluded that they are not needed. Policy addition is performed by parsing a WS-Policy4MASC file. Policy deletion is not strictly necessary because policy deactivation has similar effects. Maybe some additional policy adaptations will be considered for a future version of WS-Policy4MASC and, if needed, added into it. Our support for policy adaptation further differentiates WS-Policy4MASC from contract-based related works, such as WSOL. It is important to note that in our MASC implementation the adaptation (change) will be performed only for the policy alternative that is currently used by the process instance for which this action is executed. However, this adaptation will be also visible to all process instances using this policy alternative.

Figure 39 shows (in the Altova XmlSpy notation) the structure of the **PolicyAdaptation element type**. It extends MASCConstruct by adding 1 to many child elements that are references to any other MASC policy assertion (i.e., an instance of a sub-type of the MASCPolicyAssertion element type). These reference the policies (possibly more than 1) that are adapted. Note that PolicyAdaptation is intended to be used only as an abstract super-type for element types that describe particular versions of policy adaptation.

The definition of this element type (along with the elements <*PolicyAdaptationRef*>,) is in the file ws-policy4masc-ap.xsd , which is listed in **Appendix 6**.

Figures 40, 41, and 42 show (in the Altova XmlSpy notation) the structure of the **ActivatePolicyAssertions, DeactivatePolicyAssertions, and ChangePolicyAssertionPriority element types**. The first 2 are very simple – they extend PolicyAdaptation without adding any child element or attribute. The action that is to be performed (activation or deactivation) is determined based on the name of

the WS-Policy4MASC construct. (We have also considered an alternative design – one action with a Boolean attribute determining activation or deactivation, but chose the current design due to understandability.) ChangePolicyAssertion also inherits PolicyAdaptation, but adds 1 optional attribute (and no child elements). The positive integer attribute "NewPriority" specifies the new priority being set. Its default value is 1 (which is also the default value for policy assertion priorities).

The definitions of these element types (along with the elements <*ActivatePolicyAssertions*>, <*DeactivatePolicyAssertions*>, <*ChangePolicyAssertionPriority*>, <*ActivatePolicyAssertionsRef*>, <*DeactivatePolicyAssertionsRef*>, and <*ChangePolicyAssertionPriorityRef*>) are in the file ws-policy4masc-ap.xsd , which is listed in **Appendix 6**.

Relationship to the determined requirements for a policy language: These solutions are towards satisfying the requirements 13 "Support for Dynamic Adaptation" and (in a quite general way) 22 "Support for Dynamic Contract Manipulation".

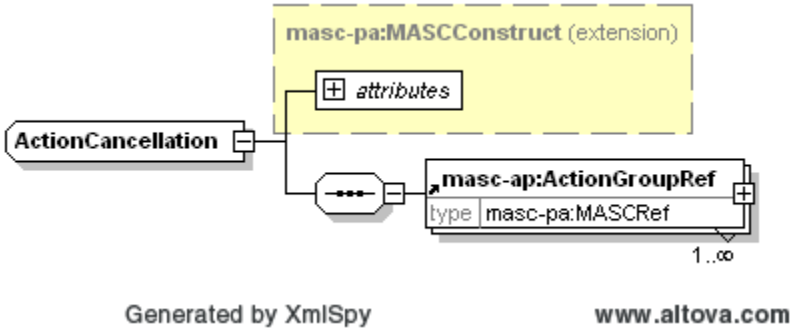

Figure 43. The structure of the ActionCancellation type

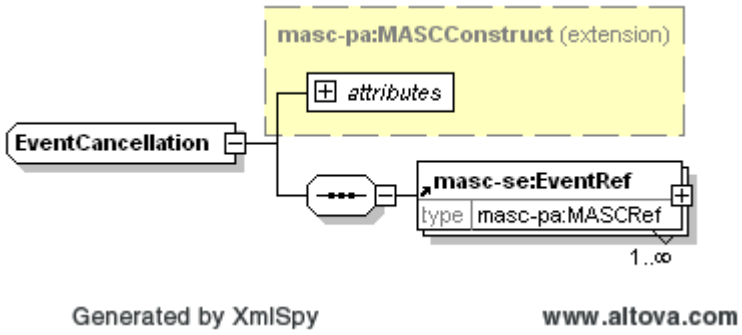

Figure 44. The structure of the EventCancellation type

ActionCancellation and EventCancellation:

It is sometimes necessary to cancel scheduled future actions. An example is when several retries of a current activity were scheduled, but only a few of them were performed and now the rest should be canceled. Event cancellation is somewhat similar to action cancellation. It makes that a previously raised event is no longer considered for triggering policy assertions. For example, after some actions (or some time) an event becomes no longer relevant, so it should no longer trigger any policy assertions. Both action cancellation and event cancellation are supported by this version of WS-Policy4MASC.

Figures 43 and 44 show (in the Altova XmlSpy notation) the structure of the **ActionCancellation and EventCancellation element types**. They both extend MASCConstruct by adding 1 to many child elements that are instances of MASCRef. In case of ActionCancellation, these are references to action groups of the canceled actions, while in the case of EventCancellation, these are references to events that are canceled.

The definitions of these element types (along with the elements <*ActionCancellation*>, <*EventCancellation*>, <*ActionCancellationRef*>, <*EventCancellationRef*>) are in the file ws-policy4masc-ap.xsd , which is listed in **Appendix 6**.

Relationship to the determined requirements for a policy language: These solutions are towards satisfying the requirement 13 "Support for Dynamic Adaptation".

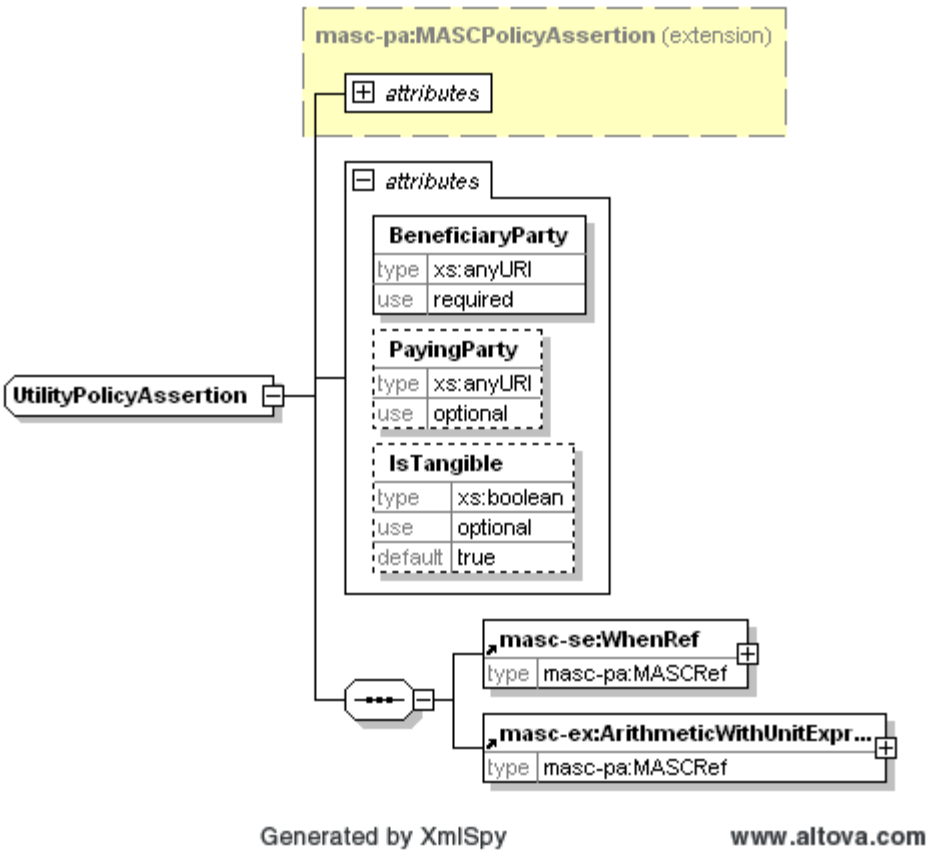

Figure 45. The structure of the UtilityPolicyAssertion type

Utility policy assertions:

A particular novelty of WS-Policy4MASC are utility policy assertions and their use (together with meta-policy assertions) for policy-based business-driven dynamic adaptation (and, in general, management) of Web service compositions. A **utility policy assertion** specifies a monetary amount assigned to a particular run-time situation. It is specified in the **<UtilityPolicyAssertion> element**, which is an instance of the UtilityPolicyAssertion element type.

After a thorough research, we have decided to differentiate between tangible value/utility (real agreed and exchanged between participants in a Web service composition) and intangible value/utility (perception about "goodness" of particular states or control actions). The motivation is that intangible value/utility is often neglected, yet it is important for business-driven decision making. In the current version of WS-Policy4MASC, all intangible values are quantified in monetary units for easier decision-making. (In a future version of WS-Policy4MASC we might consider other representations.) Monetary value is a double number with a (currency) unit – positive number means a gain to the beneficiary party, while negative number means a loss (or penalty payment from the 'paid' party to the 'payer' party). Value can be specified as a constant or as a general expression (the latter enables pay-per-volume and several other payment modes). Note that if some values in the expression have different currency units, a currency conversion Web service (specified with the ontological definition of currencies) is called. For simplicity, our MASC prototype (and, thus, our examples in WS-Policy4MASC) will only support 1 currency.

Utility policy assertions enable providing monetary amounts to <*When*> constructs that contain information about allowed states, trigger events, and optional filtering conditions. For example, it is possible to specify amounts paid when goal policy assertions are met (or not met) or when action policy assertions are executed. Since all real monetary transactions require two parties, two attributes enable specification of the beneficiary party and the paying party. A positive monetary amount means that the beneficiary party receives payment from the paying party, while a negative amount denotes that the beneficiary party has to pay the paying party. In the former case, the beneficiary party has a benefit, while in the latter it has a cost. This is inverse for the paying party. The sum of (positive) benefits and (negative) costs is a profit. While it is mandatory to specify a beneficiary party, specification of a paying party is optional. If it is missing, this means that the specified monetary amount is not a real scheduled payment, but an estimate of some possible future business value from one or several paying parties. An additional Boolean attribute specifies whether

the given amount is tangible (i.e., a real monetary amount paid) or intangible (i.e., a monetary estimate of some other business value, such as customer satisfaction). In a future version of WS-Policy4MASC, it will be possible to also specify schedules of future payments (currently, only one payment is specified per utility policy assertion), probability that an estimated future business value will be realized, and confidence in the precision of monetary estimates of intangible business values. It is also possible that a separation between various intangible business values will also be supported.

Figure 45 shows (in the Altova XmlSpy notation) the structure of the **UtilityPolicyAssertion** element type. This element type extends MASCPolicyAssertion and adds 2 required child elements, 1 required attribute and 2 optional attributes. The first child element is a MASCRef reference to a <*When*> element that specifies when the calculation (i.e. accounting) of this utility value should occur. The second child element is a MASCRef reference to a <*ArithmeticWithUnitExpression*> element that contains the expression that, when calculated, results in the monetary value. The required attribute is "BeneficiaryParty" (of data type anyURI) that is used for specification of the beneficiary party. On the contrary, the attribute "PayingParty" (also of data type anyURI) is optional. As mentioned above, if it is specified the monetary value is real payment, while if it is not specified the monetary amount is an estimate of future payments. The optional Boolean attribute "IsTangible" (default is "true") determines whether this is a tangible monetary payment or an estimate of intangible business value.

The definition of this element type (along with the elements <*UtilityPolicyAssertion*> and <*UtilityPolicyAssertionRef*>) is in the file ws-policy4masc-up.xsd , which is listed in **Appendix 7**.

Relationship to the determined requirements for a policy language: These solutions are towards satisfying the requirement 12 "Support for Specification of Monetary Implications".

A general overview of supported conflict-resolution meta-policy assertions:

Meta-policies are policies that describe other policies. Several types of meta-policies have been researched in the policy literature. However, the only type supported in WS-Policy4MASC are **conflict resolution meta-policy assertions** that specify information that helps in deciding what to do when more than one conflicting action policy assertion applies.

Such a meta-policy assertion lists several conflicting action policy assertions and a conflict resolution strategy when some of them are triggered simultaneously. At this time, we have focused on strategies that choose only the best (as defined under some criteria) among the listed alternatives. In the future, we will also research strategies that allow conditional execution of more than one alternative. The WS-Policy4MASC language is extensible, so one can add new strategies. We have defined several strategies using various maximizations of business values. Further, we have designed MASC middleware support (e.g., algorithms and data structures) for these strategies and started implementing them in our MASC prototype (the MASCPolicyDecisionMaker module). The strategies are classified along **4 mutually orthogonal dimensions**:

1. **'Only immediate' vs. 'long-term'**: For all strategies, it is possible to specify an optional ending event until which monetary values are added. If it is not specified, only immediate monetary implications of actions specified in the listed action policy are added. However, if such an ending event is specified, then a discrete event simulation is started and monetary implications of actions that are triggered by executing previously simulated actions are also calculated and added, until the specified ending event (or some specified limit of the number of counted values) is reached. Note that in the current MASC prototype, such discrete event simulations are not yet implemented, but they are planned for near future work.
2. **'Both agreed payments and estimated future business values' vs. 'only agreed payments'**: One group of strategies adds all agreed payments and estimated future business value and compares the results between alternative action policy assertions to choose only the best one. Another group of strategies adds only agreed payments, but if difference between two (or more) such sums is less or equal some specified amount, then the strategy separately adds estimated future business values and uses this as a tiebreaker. In the future, we will also develop strategies that will take into consideration probability of estimated future business values.
3. **'Both tangible and intangible' vs. 'only tangible' vs. 'only intangible'**: One group of strategies adds all tangible and intangible business values and compares the results between alternative action policy assertions to choose only the best one. Another group of strategies adds only tangible business values, but if difference between two (or more) such sums is less or equal some specified amount, then the strategy separately adds intangible business values and uses this as a tiebreaker. Conversely, yet another group of strategies adds only intangible business values, but if difference between two or more such sums is less or

equal some specified amount, then the strategy separately adds tangible business values and uses this as a tiebreaker. The latter group of strategies is used when market share, customer retention, customer satisfaction and/or other intangible business metrics are more important than immediate profit. In the future, we will also develop strategies that will take into consideration confidence in the precision of monetary estimates of intangible business values.

4. **'Benefits and costs' vs. 'cost limit'**: Here, benefits, costs, and profit are from the viewpoint of the beneficiary party. Currently, all our strategies are in the former group because we add all positive and negative monetary values and choose an alternative with the highest total profit. However, in some cases alternatives with too high costs are not acceptable (e.g., due to a lack of current funds) even if they bring higher long-term profit, so the 'cost limit' group of strategies seems useful.

Since these 4 dimensions are mutually orthogonal, a strategy specifies behavior along each dimension, e.g., 'only immediate' (dimension 1), 'only agreed payments' (dimension 2), and 'only tangible' (dimension 3). This produces 2*2*3*2=24 combinations. For the 8 combinations with 'only agreed payments' along dimension 2 and either 'only tangible' or 'only intangible' along dimension 3, it is also necessary to specify which dimension is used as the first tiebreaker, which produces 2 variations per combination. This means that, currently, the total number of strategies that we have defined is 24+8=32.

Note that WS-Policy4MASC also supports the concept of policy assertion priority, which is often used by other authors for conflict-resolution. However, in our work **business value is more important than policy assertion priority**, which is an optional feature (and not used in our current examples). That is, ONLY if two action policy assertions produce the same business value (according to the specified conflict-resolution strategy) policy assertion priorities are considered (if they are specified at all). This differentiates our research from many related works.

Relationship to the determined requirements for a policy language: These solutions are towards satisfying the requirements 12 “Support for Specification of Monetary Implications” and 13 “Support for Dynamic Adaptation”.

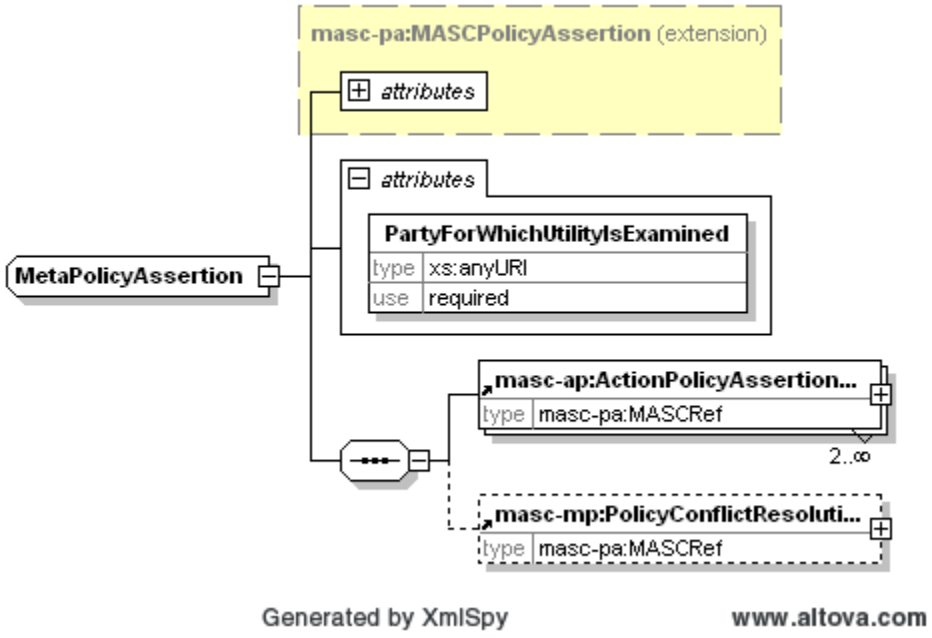

Figure 46. The structure of the MetaPolicyAssertion type

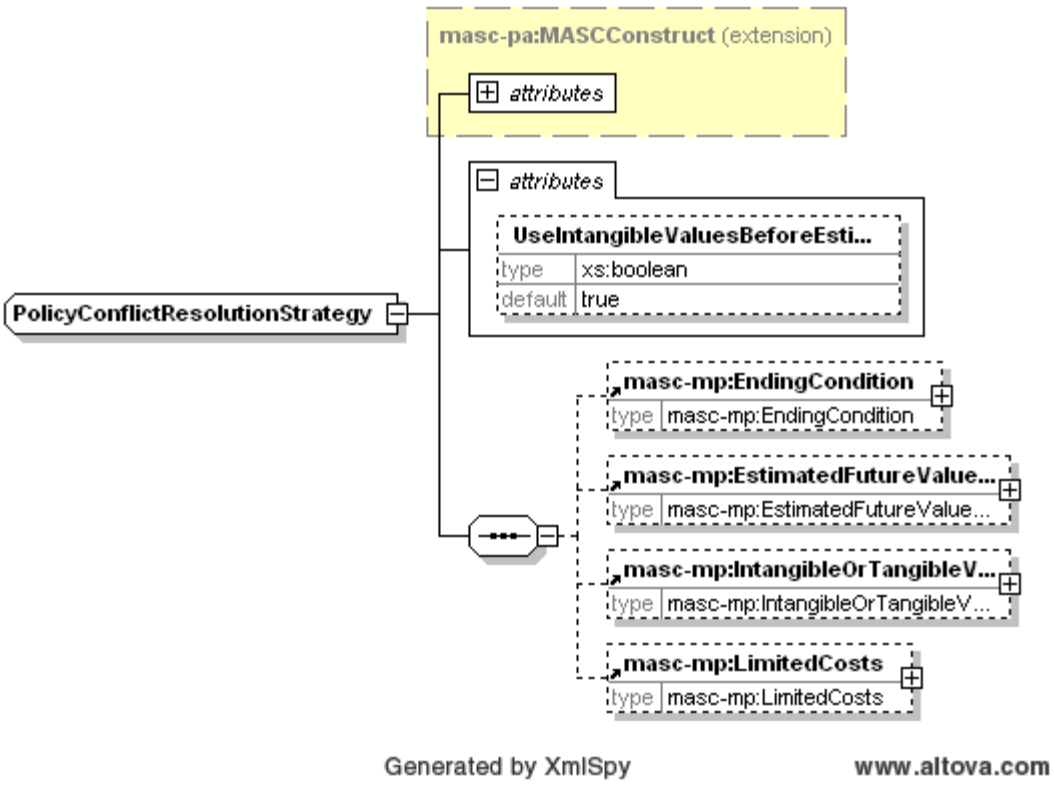

Figure 47. The structure of the PolicyConflictResolutionStrategy type

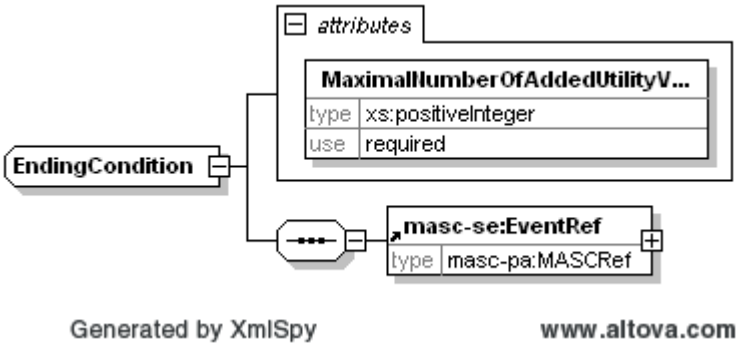

Figure 48. The structure of the EndingCondition type

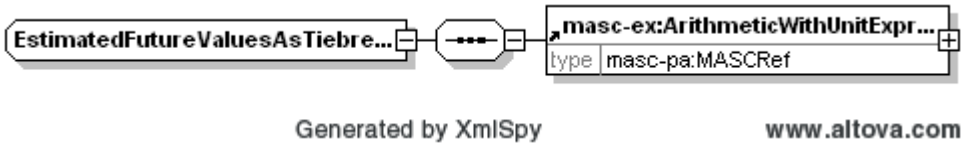

Figure 49. The structure of the EstimatedFutureValuesAsTiebreakerOnly type

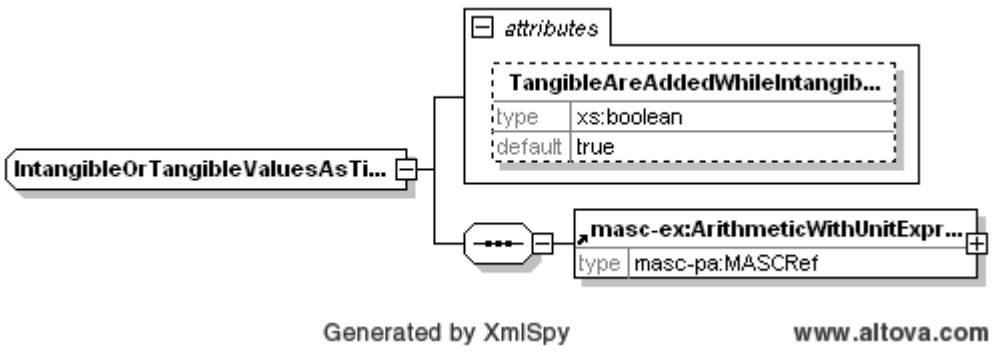

Figure 50. The structure of the IntangibleOrTangibleValuesAsTiebreakerOnly type

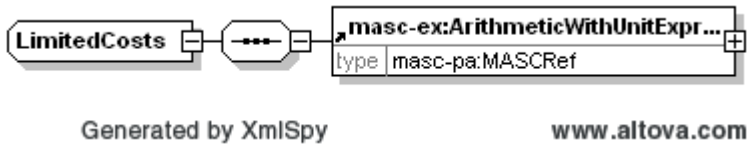

Figure 51. The structure of the LimitedCosts type

MetaPolicyAssertion, PolicyConflictResolutionStrategy, and related types:

Figure 46 shows (in the Altova XmlSpy notation) the structure of the **MetaPolicyAssertion** element type. This element type extends MASCPolicyAssertion and adds 2 to many required child elements, 1 optional child element, and 1 required attribute. The required child elements (at least 2) are MASCRef references to action policy assertions. The optional child element is a MASCRef reference to a policy conflict resolution strategy (specified in a <PolicyConflictResolutionStrategy> construct). If no policy conflict resolution strategy is specified or it produces several equally valid options, then only the highest priority action policy assertion is executed (if priorities are specified at all). If even after this there are several equally valid options, then one of them is chosen at random (i.e., it is not determined which one will execute). The required attribute "PartyForWhichUtilityIsExamined" (of the data type anyURI) specifies from the viewpoint of which party the utility is calculated.

The definition of this element type (along with the elements <*MetaPolicyAssertion*> and <*MetaPolicyAssertionRef*>) is in the file ws-policy4masc-mp.xsd , which is listed in **Appendix 8**.

Figures 47 to 51 show (in the Altova XmlSpy notation) the structure of the **PolicyConflictResolutionStrategy, EndingCondition, EstimatedFutureValuesAsTiebreakerOnly, IntangibleOrTangibleValuesAsTiebreakerOnly, and LimitedCosts element types**. The last 4 are used to represent the above-mentioned 4 dimensions of business-driven policy assertion conflict resolution. PolicyConflictResolutionStrategy extends MASCConstruct by adding 4 optional elements (1 per each of the 4 dimensions) and 1 required attribute (which determines precedence between dimensions 2 and 3). Since meta-policies were discussed above, we will not provide a detailed discussion of each element and attribute of these 5 types here. (This will be given in a future version of this document.)

The definition of these element types (along with the instance elements and references) is in the file ws-policy4masc-mp.xsd , which is listed in **Appendix 8**.

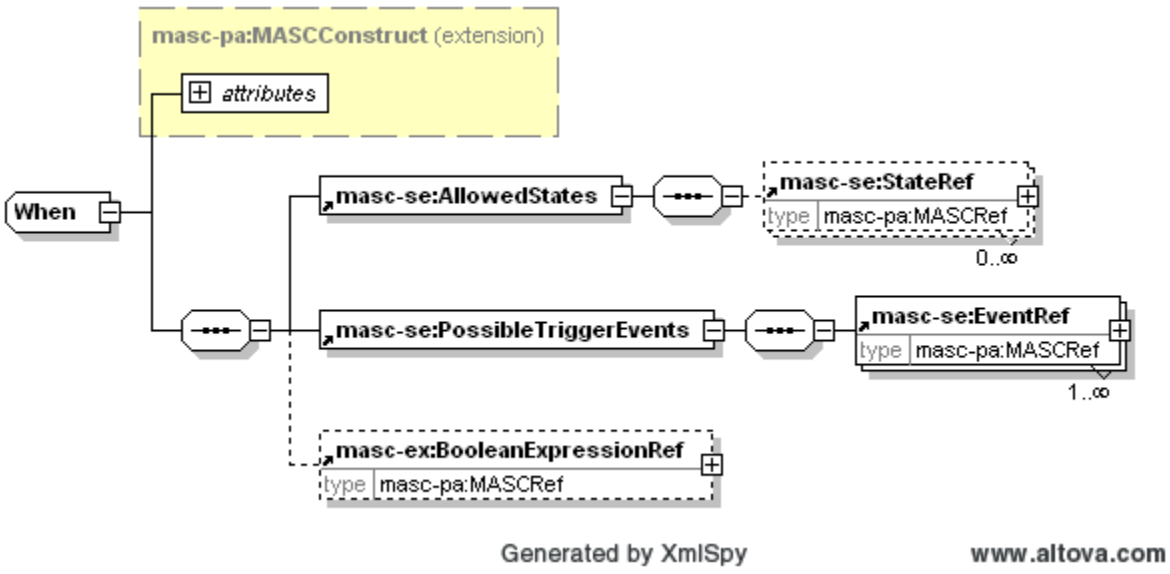

Figure 52. The structure of the When type

The When element type:

The **When element type** specifies the conditions (states, trigger events, optional additional conditions) that have to be satisfied to execute the actions. Notice that the condition specified within the <When> element

contains a Boolean expression (which might contain other expressions), that, if "true", triggers processing of the WS-Policy4MASC construct (e.g., policy assertion) that references it. <When> is modeling a complex precondition, which contains a state in which the system has to be, a list of events that can occur, and additional (pre)condition to be satisfied (e.g., used for filtering of events). We gave it the name "When" instead of the "Precondition" because it is more logical in some situations to say "When … Do …" and because <When> is used in some situations (e.g., state transition) where the term "Precondition" is not fully appropriate. It extends MASCConstruct. For its structure, see **Figure 52** and file ws-policy4masc-se.xsd (**Appendix 9**). See **Appendices 1, 2, and 3** for several examples.

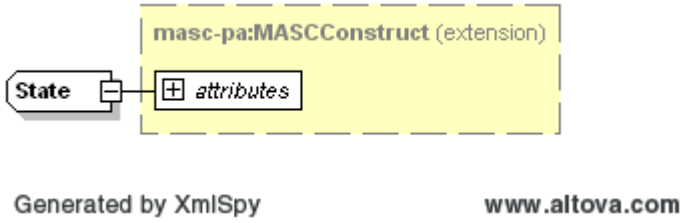

Figure 53. The structure of the State type

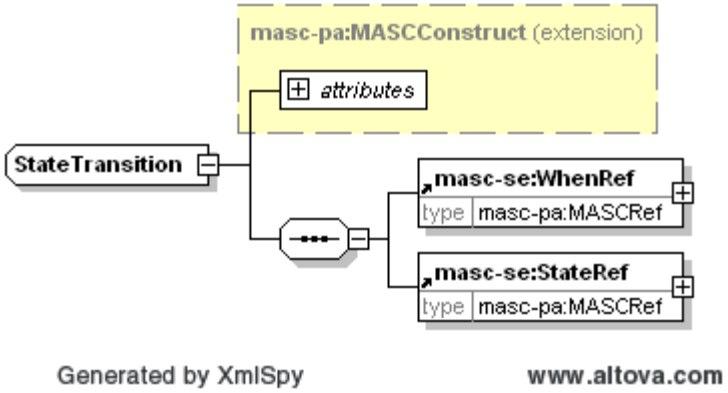

Figure 54. The structure of the StateTransition type

State and StateTransition:

The **State element type** is used to define states (more precisely: IDs of states). It extends MASCConstruct. It is important because separation between monitoring (regular execution) and control (adaptation) is achieved through the concept of a (health) state. Many different events can map into the same state, so using states helps maintainability and understandability of policy assertions. For its structure, see **Figure 53** and file ws-policy4masc-se.xsd (**Appendix 9**). See **Appendices 1, 2, and 3** for several examples.

The **StateTransition element type** defines state transitions. It contains a reference to a <When> element and a reference to the new state (the references to the possible old states are in the <When> element). For its structure, see **Figure 54** and file ws-policy4masc-se.xsd (**Appendix 9**).

Relationship to the determined requirements for a policy language: The WS-Policy4MASC concepts of a state and a state transition are towards satisfying the requirement 14 "Specification of Health States".

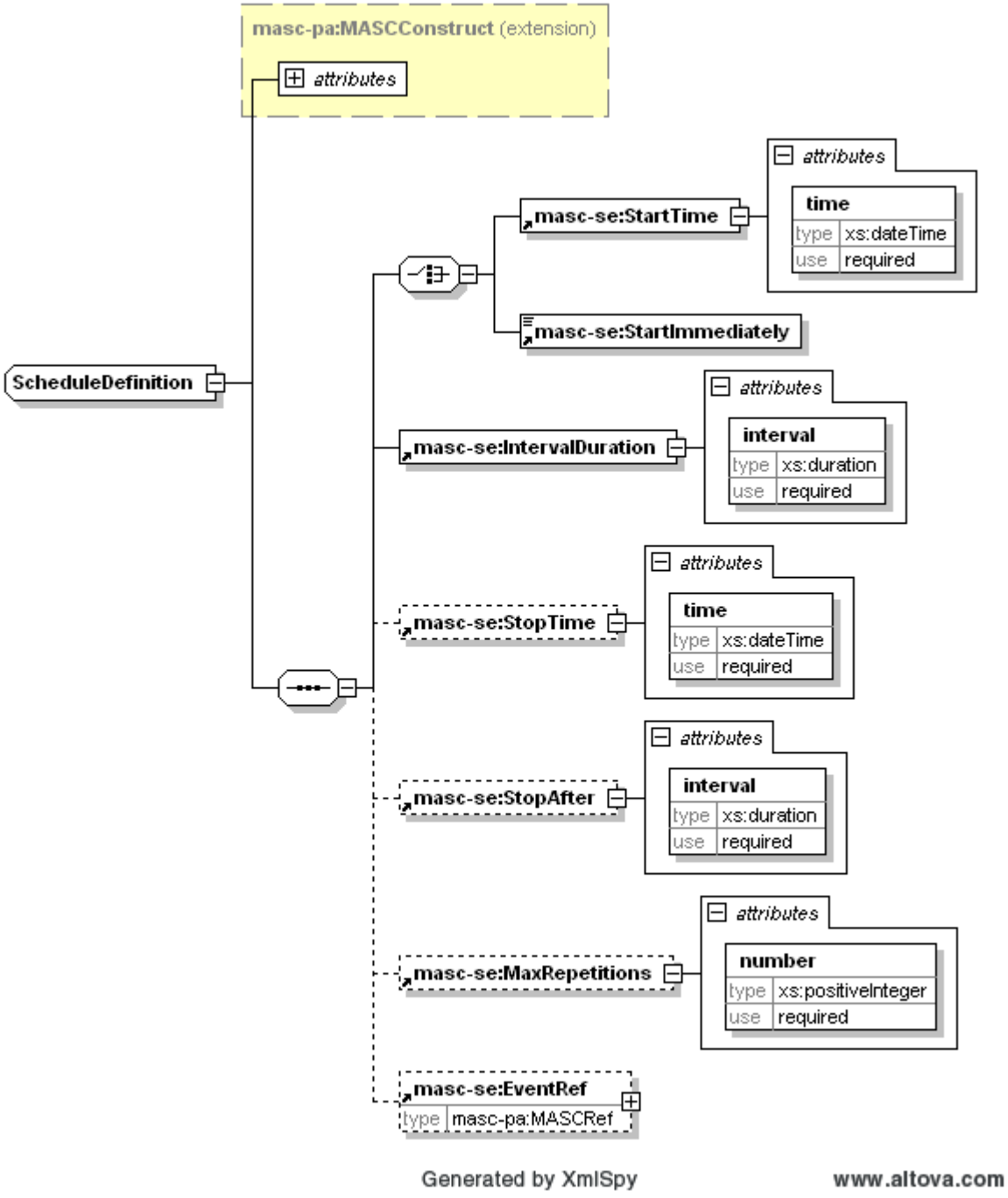

Figure 55. The structure of the ScheduleDefinition type

ScheduleDefinition:

The **ScheduleDefinition element type** is used to define various schedules. There are many periodic activities that can be specified using this element. It contains information (elements) about the schedule start time (particular date/time, immediately when the schedule is encountered, or after some waiting period), interval duration, and information about how the schedule execution is ended. A schedule can be ended at particular stop date/time, after some period, after a maximum number of repetitions is made, or when some event happens. Consequently, the ScheduleDefinition element type contains optional elements for all these stop criteria. If more than one stop criteria are specified, the schedule is stopped when the first one occurs. If no stop criterion is specified, the schedule is executed until a policy assertion is changed. Here is an illustrative example. Assume that we need to specify that backup must be done every night at 2:00AM. WS-Policy4MASC enables this by defining a schedule (in a <*ScheduleDefinition*>) that starts at 2:00AM one particular day, has interval of 1 day, and never stops. Then, it is defined (in an <*EventDefinition*>) that an event is raised whenever the time specified in this schedule is reached. Then, it is specified (within an

<*ActionPolicyAssertion*>) that the backup action should be performed whenever this event happens. Another example is to calculate average response time every midnight as an average of response times measured during the last day. ScheduleDefinition extends MASCConstruct. For its structure, see **Figure 55** and file ws-policy4masc-se.xsd (**Appendix 9**).

Figure 56. The structure of the EventDefinition type

EventDefinition:

The **EventDefinition element type** is used to specify events. The current WS-Policy4MASC grammar supports specification of many different event types, such as update of a monitored data item, operation invocation, sending or receipt of a message, satisfaction of a goal policy assertion, execution of an action policy assertion, occurrence of a previously scheduled periodic event, start or finish of a process instance, initialization or finalization of Web service, or state entrance/exit/transition. ScheduleDefinition extends MASCConstruct. For its structure, see **Figure 56** and file ws-policy4masc-se.xsd (**Appendix 9**). Note that for a number of possible events have special attributes and/or child elements. For these details, examine the

file ws-policy4masc-se.xsd (**Appendix 9**), as well as the scopes defined in the file ws-policy4masc-sc.xsd (**Appendix 10**). Note that the specification of events is to be significantly improved in the next version of the WS-Policy4MASC language. See **Appendices 1, 2, and 3** for several examples.

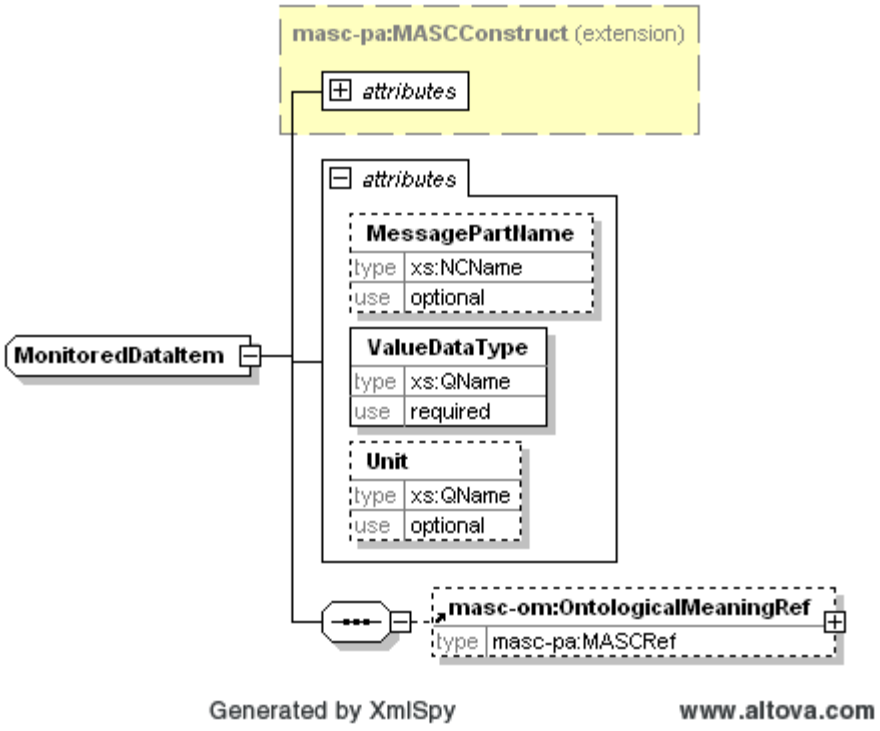

Figure 57. The structure of the MonitoredDataItem type

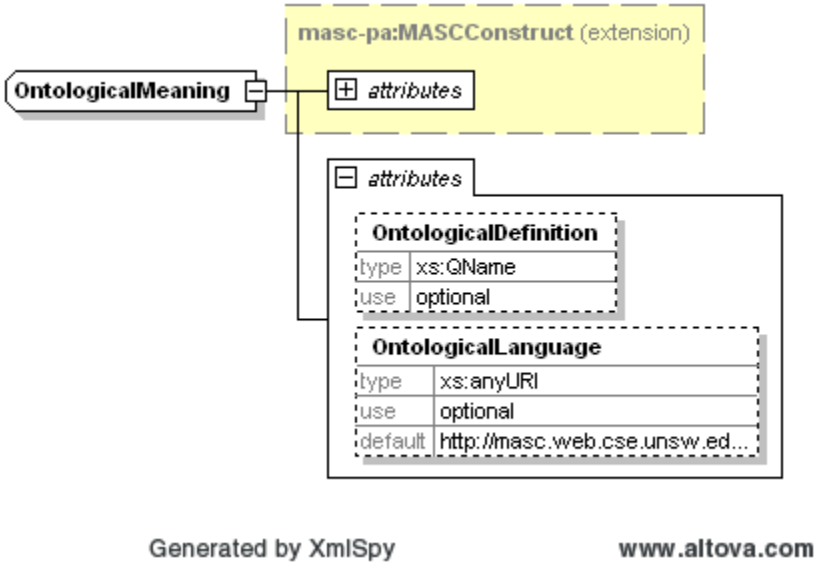

Figure 58. The structure of the OntologicalMeaning type

MonitoredDataItem and Ontological Meaning:

The **MonitoredDataItem element type** is used to describe which data items (e.g., QoS metrics or SOAP message part) should be monitored. If there is a value in the optional parameter "MessagePartName", then this is description of a SOAP message part. Otherwise, this is a description of a QoS metric. In addition to general purpose modules for calculation/evaluation of expressions, the MASC middleware will also contain some specialized modules using monitored data item definitions to perform measurement of time and number of invocations (these are basic QoS metrics from which many other can be calculated, e.g., response time, throughput, availability, reliability). While these modules will be designed in the overall MASC

middleware architecture, their implementation is not a priority for our research. The attributes "ValueDataType" and "Unit" specify data type and unit of measurement (the later is relevant only for QoS metrics). In addition, there is a optional reference to an ontological meaning of the data item. For the structure of the MonitoredDataItem element type see **Figure 57** and file ws-policy4masc-gp.xsd (**Appendix 5**). See **Appendices 1, 2, and 3** for several examples.

The definition of ontological meaning in the current version of WS-Policy4MASC is rudimentary and it is contained in the **OntologicalMeaning** element type. At this time, ontological meaning is used only for comparisons. (In principle, it can be used as configuration data for monitoring/calculation, but this might be quite complicated in practice.) OntologicalMeaning has 2 optional attributes: "OntologicalDefinition" is a qualified XML name containing namespace of the used ontology and name of the ontological concept within this ontology, while"OntologyLanguage" is the URI of the language in which the referenced ontology is defined. If a Web service does not understand this ontology language, it can perform simple syntax matching of ontological definitions. The actual definitions of ontological concepts are in external, reusable and extensible, ontologies. We allow that these ontologies can be specified in any of the currently used ontology languages, such OWL, RDF, RDF Schema, or XML Schema. However, this causes that interoperability suffers and the requirement of using minimal number of Web service languages is not satisfied. Therefore, we have also provided a very simple ontology schema (first used for WSOL) that is a default ontology language. This simple format is not enough to support ontological reasoning. For the structure of the MonitoredDataItem element type see **Figure 58** and file ws-policy4masc-op.xsd (**Appendix 11**). See **Appendices 1, 2, and 3** for several examples.

A note about specification of expressions in WS-Policy4MASC:

For the specification of expressions, we have first decided to leverage and adapt the WSOL expression XML schema. A few changes were needed to make it applicable to the WS-Policy4MASC context. However, this expression grammar is very powerful and maybe it is likely that in early prototypes of the MASC middleware it will be supported only partially.

In the earlier versions of the WS-Policy4MASC language, we have used this modified WSOL expressions XML schema. However, it is monolithic, so to achieve modularity we have to break it into smaller parts. To achieve reusability analogously to the other parts of WS-Policy4MASC, MASCRef references have to be used. Therefore, we have made a decision to redesign the specification of expressions in WS-Policy4MASC starting from the basics and then gradually adding powerful concepts from the modified WSOL expressions XML schema.

Unfortunately, this process is not yet finished. Consequently, at this time we have two expression schemas in WS-Policy4MASC: a) the one from the previous version is powerful, but monolithic (and even causes a few syntax errors, which can be fixed easily if needed); b) the new one that we started developing is flexible, but at this time in a very simple form. Of course, this situation is not satisfactory and providing an appropriate specification of expressions (again) is now our top priority (actually, we are already working on it). For your information, both versions of the file ws-policy4masc-ex.xsd are given in **Appendix 12**.

Policy attachment:

WS-Policy4MASC policy assertions can refer to (and be attached to) WSDL constructs (e.g., Web services, endpoints/ports, operations, messages, message parts), and WS-BPEL/XAML constructs (e.g., process, sub-process, activity). The current version of the WS-Policy4MASC language completely relies on WS-PolicyAttachment (a mechanism of WS-Policy) for association (attachment) of policies (and their elements) to policy domains, i.e., Web services and Web service compositions to which they apply. At this time, this seems as a satisfactory solution, which works in all situations we have encountered so far. However, additional (or maybe even alternative) solutions will be developed if during the future development of WS-Policy4MASC it turns out that the WS-Policy mechanisms cannot handle some situations.

Relationship to the determined requirements for a policy language: These architectural decisions are towards satisfying the requirements 2 "Compatibility with Industry Standards for Web Services" and 3 "Compatibility with Microsoft Solutions for Web Services".

On complexity of the language constructs:

Even from this relatively brief discussion of an incomplete version of the WS-Policy4MASC language,

someone could conclude that the language is too complicated and too verbose. This is probably true, but only partially. The fact is that the language is ambitious and that it tries to address many operations (particularly, QoS monitoring, various dynamic adaptation mechanisms) of a policy-based middleware for Web services and Web service compositions. However, we have tried to make the language modular and extensible. While the language will grow in the future (both in the number of core concepts and, particularly, in terms of extensions of these core concepts), we will also actively explore possibilities for streamlining, both by reducing the number of core concepts (e.g., by moving some core concepts into optional extensions) and by reducing the number of elements in the XML Schema grammar of the language (e.g., by having general elements that are distinguished by a 'type' or 'xsi:type' attribute instead of having special elements). However, there are tradeoffs associated with this. We can illustrate them on the example of using attributes instead of elements. First, an object model generated from the schema that contains elements is bigger (due to automatic generation of C# classes from XML schemas this size should not be an issue), but it is easier and faster to dynamically find appropriate information in it. Also, elements are more extensible than attributes and they can contain a bit more metadata than attributes.

Relationship to the determined requirements for a policy language: Our primary concern in these decisions was to satisfy the identified policy language requirements, which are diverse and demanding. For example, part of the complexity comes from the need to provide precision, as specified in the requirement 1 “Formality, Precision, XML Encoding”. Satisfaction of the requirements 23 “Reusability Constructs” and 24 “Automatic Code Generation” also significantly increases complexity.

VI) Current Organization of WS-Policy4MASC Grammar into XML Schemas

One of the most visible changes introduced since WS-Policy4MASC version 0.6 is that the language grammar is broken into several smaller XML schemas (i.e., separate XML Schema files). As the language grows, such modularization significantly helps in understandability and maintainability of the language grammar.

The current XML schemas are given in the Appendices 4 to 12. Note that conventions for formatting (e.g., names) were used. For example, all file names have the same format "ws-policy4masc-XY-AB.xsd" where XY are two letters that characterize the contents of the schema (e.g., "gp" for goal policy assertions, "se" for states and events) and AB are two digits (without a period between them) for the language version (e.g., "08"). Additionally, all namespaces use analogous format "http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-XY" (e.g., "http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-gp"). In the next version of WS-Policy4MASC, the URI http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc (notice the error: 'sce' should have been 'cse') should be replaced with the URI http://msc.web.sce.unsw.edu.au/ws-policy4masc as this is the official MASC website.

Unfortunately, our version of Altova XmlSpy for some reason destroyed our nice formatting of the files. In particular, we have found some unexpected problems with ordering of comments (e.g., our comments for attributes are usually misplaced!), indentation, and line breaks. We have tried to manually correct some problems, but did not have time to correct all of them. We apologize sincerely.

The files listed in Appendices 4 to 12 are:

4) **ws-policy4masc-pa-08.xsd**: Contains grammar for the definition of MASCConstruct, MASCRef, and MASCPolicyAssertion.
5) **ws-policy4masc-gp-08.xsd**: Contains grammar for the definition of goal policy assertions and monitored data items.
6) **ws-policy4masc-ap-08.xsd**: Contains grammar for the definition of action policy assertions and details for various process adaptation actions that can be specified in action policy assertions.
7) **ws-policy4masc-up-08.xsd**: Contains grammar for the definition of utility policy assertions.
8) **ws-policy4masc-mp-08.xsd**: Contains grammar for the definition of meta-policy assertions and conflict resolution strategies.
9) **ws-policy4masc-se-08.xsd**: Contains grammar for the definition of When constructs, states, state transitions, schedules, and events (including details of many pre-defined events).
10) **ws-policy4masc-sc-08.xsd**: Contains grammar for the definition of scopes (service, endpoint, operation, message, and message park) and corresponding constants.
11) **ws-policy4masc-om-08.xsd**: Contains grammar for the definition of ontological meaning.
12) **ws-policy4masc-ex-08.xsd**: Contains grammar for the definition of various expressions.

References

NOTE: Many additional references can be found in our academic papers.

[K&W04] Kephart, J.O., Walsh, W.E. (2004), "An Artificial Intelligence Perspective on Autonomic Computing Policies", in Proceedings of Policy 2004, IEEE, pp. 3-12.

[LDK04] Ludwig, H., Dan, A. Kearney, R. (2004), "Cremona: An Architecture and Library for Creation and Monitoring of WS-Agreements", in Proceedings of ICSOC'04, ACM, pp. 65-74.

[TLT05] Tosic, V., Lutfiyya, H., Tang, Y. (2005), "Major Requirements for a System for Comprehensive Management of XML Web Services", *Technical Report #644*, Department of Computer Science, University of Western Ontario, April 2005.

[TLT06] Tosic, V., Lutfiyya, H., Tang, Y. (2006), "Web Service Offerings Language (WSOL) Support for Context Management of Mobile/Embedded Web Services", in Proceedings of ICIW'06, IEEE-CS.

[TPP05] Tosic, V., Pagurek, B., Patel, K., Esfandiari, B., Ma, W. (2005), "Management Applications of the Web Service Offerings Language (WSOL)", Information Systems, Elsevier, Vol. 30, No. 7, pp. 564-586.

[WSP04] Schlimmer, J. (ed.) (2004), "Web Services Policy Framework (WS-Policy), version: Sept. 2004", on-line document at: www6.software.ibm.com/software/developer/library/ws-policy.pdf

[WSP06] W3C Web Services Policy Working Group (2006), "Web Services Policy (WS-Policy), version: 1.5, Nov. 2006, on-line document at: www.w3.org/TR/ws-policy/

Appendix 1: The Example File masc-gp-08-Example2-output.xml

A weather report Web service has one operation: Integer weatherTemperature(String postalCode). This appendix shows how WS-Policy4MASC can be used to specify the post-condition that the result represents temperature in Celsius degrees and that it should be between -70C and 50C. WS-Policy4MASC policy assertions and other constructs are specified within a WS-Policy element. First, the <MonitoredDataItem>, <MonitoredDataItemCollection>, and <ActionGroup> constructs specify that the message part "weatherTemperature" is monitored and expressed in Celsius degrees. (For the above-mentioned reasons, this specification is verbose.) Definitions of states and event follows. Then, a <When> construct referring to the state "Executing" and the event "MessageToBeSent" is defined. The subsequent action policy assertion specifies that when this event happens in this state, monitoring of the message part "weatherTemperature" is performed by the provider Web service. This action policy assertion is used to configure MASC monitoring modules. Boolean expression "LimitsOfValidWeatherTemperature" is a complex expression that specifies that values of the monitored data item (the message part "weatherTemperature") must be between -70C and 50C. The above-mentioned <When> construct and Boolean expression are referenced in the definition of the subsequent goal policy assertion, which is also used to configure MASC monitoring modules. This goal policy assertion specifies that when the event "MessageToBeSent" occurs in the state "Executing", then the provider Web service should evaluate the mentioned Boolean expression. It also states that the provider is responsible for meeting this goal. In a separate file (omitted from these Appendices), a WS-Policy policy attachment element defines that this defined policy is applied to the reply message of the current weather report Web service.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!-- Example 2 of GoalPolicyAssertion schema (and others) in WS-Policy4MASC version 0.8
        This example is for the Canadian Weather Report example.
        The only Web service operation is: Integer weather(String postalCode).
        This example is for the output message - the policy must be attached to this message.
-->
<!--Copyright (c) 2006 The University of New South Wales -->
<!-- NOTE: The following element should also contain the attribute:
wsu:Id="CanadianWhetherReport-Output"
-->
<wsp:Policy xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
xmlns:masc-gp="http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-gp"
xmlns:masc-se="http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-se"
xmlns:masc-ex="http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-ex" xmlns:masc-
om="http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-om"  xmlns:masc-
sc="http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-sc" xmlns:masc-
ap="http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-ap"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://schemas.xmlsoap.org/ws/2004/09/policy C:\workspace\WS-Policy4MASCv0-
8\Schemas\ws-policy.xsd http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-gp C:\workspace\WS-
Policy4MASCv0-8\Schemas\ws-policy4masc-gp.xsd
http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-se C:\workspace\WS-Policy4MASCv0-8\Schemas\ws-
policy4masc-se.xsd
http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-ex C:\workspace\WS-Policy4MASCv0-8\Schemas\ws-
policy4masc-ex.xsd
http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-om C:\workspace\WS-Policy4MASCv0-8\Schemas\ws-
policy4masc-sc.xsd
http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-sc C:\workspace\WS-Policy4MASCv0-8\Schemas\ws-
policy4masc-om.xsd
http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-ap C:\workspace\WS-Policy4MASCv0-8\Schemas\ws-
policy4masc-ap.xsd"
targetNamespace="TARGETmasc-gp-Example2" xmlns:tns="masc-gp-Example2"
xmlns:wsdlfile="C:\workspace\WS-Policy4MASCv0-8\Examples\WSDL-Example2.xml"
xmlns:ontology1="C:\workspace\WS-Policy4MASCv0-8\Examples\Ontology-Example2.xml">
        <!--
                        Definitions of ontological meaning.
    -->
```

```
<masc-om:OntologicalMeaning MASCID="Temperature-Weather" OntologicalDefinition="ontology1:Temperature-Weather"/>
    <!--
            Definitions of monitored data items.
    -->
    <masc-gp:MonitoredDataItem MASCID="Temperature-Weather-InCelsius" MessagePartName="weatherTemperature" ValueDataType="xs:integer" Unit="ontology1:Celsius">
        <masc-om:OntologicalMeaningRef To="tns:Temperature-Weather"/>
    </masc-gp:MonitoredDataItem>
    <!--
            Definitions of states.
    -->
    <masc-se:StateDefinition MASCID="Executing"/>
    <!--
            Definitions of events.
    -->
    <masc-se:EventDefinition MASCID="ReplyToBeSent">
        <masc-se:MessageToBeSent>
            <!-- NOTE: Which message is sent could be determined by policy attachment of
            this policy. However, the current WS-Policy4MASC grammar requires specifying
            this information explicitly in the following masc-sc:Scope-Messag element, but
            this might be made obsolete due to possible conflicts with policy attachment. -->
            <masc-sc:Scope-Message>
                <masc-sc:ServiceName Name="wsdlfile:WeatherReport"/>
                <masc-sc:EndpointName Name="wsdlfile:CanadianWeatherReport"/>
                <masc-sc:OperationName Name="wsdlfile:WeatherTemperature"/>
                <masc-sc:MessageName Name="wsdlfile:WeatherTemperatureReply"/>
            </masc-sc:Scope-Message>
        </masc-se:MessageToBeSent>
    </masc-se:EventDefinition>
    <!--
            Definitions of When constructs.
    -->
    <masc-se:When MASCID="Executing-ReplyToBeSent">
        <masc-se:AllowedStates>
            <masc-se:StateRef To="tns:Executing"/>
        </masc-se:AllowedStates>
        <masc-se:PossibleTriggerEvents>
            <masc-se:EventRef To="tns:ReplyToBeSent"/>
        </masc-se:PossibleTriggerEvents>
    </masc-se:When>
    <!--
            Definitions of monitoring data collection actions.
    -->
    <masc-ap:MonitoredDataItemCollection MASCID="MonitoringOfTemperature-Weather-InCelsius">
        <masc-gp:MonitoredDataItemRef To="tns:Temperature-Weather-InCelsius"/>
    </masc-ap:MonitoredDataItemCollection>
    <!--
            Definitions of monitoring action groups.
    -->
    <masc-ap:ActionGroup MASCID="Monitoring">
        <masc-ap:MonitoredDataCollectionRef To="tns:MonitoringOfTemperature-Weather-InCelsius"/>
    </masc-ap:ActionGroup>
    <!--
            Definitions of action policy assertions configuring monitoring.
    -->
    <masc-ap:ActionPolicyAssertion MASCID="MonitorResultValue" ManagementParty="MASC_WSPROVIDER">
        <masc-se:WhenRef To="tns:Executing-ReplyToBeSent"/>
        <masc-ap:ActionGroupRef To="tns:Monitoring"/>
    </masc-ap:ActionPolicyAssertion>
    <!--
            Definitions of BooleanExpressions for goal policy assertions.
    -->
```

```
        <masc-ex:BooleanExpression MASCID="LowerLimitOfValidWeatherTemperature">
                <masc-ex:BooleanConstant>true</masc-ex:BooleanConstant>
                <!-- The following elements should be specified instead of the above
                Boolean constraint element that is always true, but this is not yet supported by the
                WS-Policy4MASC grammar (the simplified expressions schema).
                <masc-ex:ArithmeticWitUnitComparator Type="GreaterOrEqual"/>
                <masc-ex:MonitoredDataItemRef To="tns:Temperature-Weather-InCelsius"/>
                <masc-ex:ArithmeticWitUnitConstant>
                        <masc-ex:ArithmeticValue>-70</masc-ex:ArithmeticValue>
                        <masc-ex:Unit OntologicalType="ontology1:Celsius"/>
                </masc-ex:ArithmeticWitUnitConstant>
                -->
        </masc-ex:BooleanExpression>
        <masc-ex:BooleanExpression MASCID="UpperLimitOfValidWeatherTemperature">
                <masc-ex:BooleanConstant>true</masc-ex:BooleanConstant>
                <!-- The following elements should be specified instead of the above
                Boolean constraint element that is always true, but this is not yet supported by the
                WS-Policy4MASC grammar (the simplified expressions schema).
                <masc-ex:ArithmeticWitUnitComparator Type="LessOrEqual"/>
                <masc-ex:MonitoredDataItemRef To="tns:Temperature-Weather-InCelsius"/>
                <masc-ex:ArithmeticWitUnitConstant>
                        <masc-ex:ArithmeticValue>50</masc-ex:ArithmeticValue>
                        <masc-ex:Unit OntologicalType="ontology1:Celsius"/>
                </masc-ex:ArithmeticWitUnitConstant>
                -->
        </masc-ex:BooleanExpression>
        <masc-ex:BooleanExpression MASCID="LimitsOfValidWeatherTemperature">
                <masc-ex:BooleanConstant>true</masc-ex:BooleanConstant>
                <!-- The following elements should be specified instead of the above
                Boolean constraint element that is always true, but this is not yet supported by the
                WS-Policy4MASC grammar (the simplified expressions schema).
                <masc-ex:BooleanOperator Type="AND"/>
                <masc-ex:BooleanExpressionRef To="tns:LowerLimitOfValidWeatherTemperature"/>
                <masc-ex:BooleanExpressionRef To="tns:UpperLimitOfValidWeatherTemperature"/>
                -->
        </masc-ex:BooleanExpression>
        <!--
                        Definitions of goal policy assertions.
    -->
        <masc-gp:GoalPolicyAssertion MASCID="ValidWeatherTemperature"
ResponsibleParty="MASC_WSPROVIDER" ManagementParty="MASC_WSPROVIDER">
                <masc-se:WhenRef To="tns:Executing-ReplyToBeSent"/>
                <masc-ex:BooleanExpressionRef To="tns:LimitsOfValidWeatherTemperature"/>
        </masc-gp:GoalPolicyAssertion>

</wsp:Policy>
```

Appendix 2: The Example File masc-gp-08-Example3.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!-- Example 3 of GoalPolicyAssertion schema (and others) in WS-Policy4MASC version 0.8
        This example is for the Canadian Weather Report example.
        The only Web service operation is: Integer weather(String postalCode).
        This example is for the whole operation - the policy must be attached to this operation.
-->
<!--Copyright (c) 2006 The University of New South Wales -->
<!-- NOTE: The following element should also contain the attribute:
wsu:Id="CanadianWhetherReport-WholeOperation-Monitoring"
-->
<wsp:Policy xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
xmlns:masc-gp="http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-gp"
xmlns:masc-se="http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-se"
xmlns:masc-ex="http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-ex" xmlns:masc-
om="http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-om"  xmlns:masc-
sc="http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-sc" xmlns:masc-
ap="http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-ap"
xmlns:masc-up="http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-up"
xmlns:masc-mp="http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-mp"
xmlns:configData="http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/configData"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://schemas.xmlsoap.org/ws/2004/09/policy C:\workspace\WS-Policy4MASCv0-
8\Schemas\ws-policy.xsd http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-gp C:\workspace\WS-
Policy4MASCv0-8\Schemas\ws-policy4masc-gp.xsd
http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-se C:\workspace\WS-Policy4MASCv0-8\Schemas\ws-
policy4masc-se.xsd
http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-ex C:\workspace\WS-Policy4MASCv0-8\Schemas\ws-
policy4masc-ex.xsd
http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-om C:\workspace\WS-Policy4MASCv0-8\Schemas\ws-
policy4masc-sc.xsd
http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-sc C:\workspace\WS-Policy4MASCv0-8\Schemas\ws-
policy4masc-om.xsd
http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-ap C:\workspace\WS-Policy4MASCv0-8\Schemas\ws-
policy4masc-ap.xsd
http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-up C:\workspace\WS-Policy4MASCv0-8\Schemas\ws-
policy4masc-up.xsd
http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-mp C:\workspace\WS-Policy4MASCv0-8\Schemas\ws-
policy4masc-mp.xsd http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/configData C:\workspace\WS-
Policy4MASCv0-8\Schemas\configData.xsd"
targetNamespace="TARGETmasc-mp-Example1" xmlns:tns="masc-mp-Example2"
xmlns:wsdlfile="C:\workspace\WS-Policy4MASCv0-8\Examples\WSDL-Example2.xml"
xmlns:ontology1="C:\workspace\WS-Policy4MASCv0-8\Examples\Ontology-Example2.xml"
xmlns:gpExample2="C:\workspace\WS-Policy4MASCv0-8\Examples\masc-gp-08-Example2-output.xml">
        <!--
                        Definitions of ontological meaning.
    -->
    <masc-om:OntologicalMeaning MASCID="CurrentTime" OntologicalDefinition="ontology1:CurrentTime"/>
    <masc-om:OntologicalMeaning MASCID="ResponseTime" OntologicalDefinition="ontology1:ResponseTime"/>
        <!--
                        Definitions of monitored data items.
    -->
     <masc-gp:MonitoredDataItem MASCID="ResponseTimeStart-InMilliseconds"  ValueDataType="xs:integer"
Unit="ontology1:Millisecond">
                <masc-om:OntologicalMeaningRef To="tns:CurrentTime"/>
    </masc-gp:MonitoredDataItem>
     <masc-gp:MonitoredDataItem MASCID="ResponseTimeStop-InMilliseconds"  ValueDataType="xs:integer"
Unit="ontology1:Millisecond">
                <masc-om:OntologicalMeaningRef To="tns:CurrentTime"/>
    </masc-gp:MonitoredDataItem>
```

```
  <masc-gp:MonitoredDataItem MASCID="ResponseTime-InMilliseconds"  ValueDataType="xs:integer"
Unit="ontology1:Millisecond">
            <masc-om:OntologicalMeaningRef To="tns:ResponseTime"/>
  </masc-gp:MonitoredDataItem>
      <!--
                  Definitions of events.
  -->
      <masc-se:EventDefinition MASCID="RequestToBeSent">
            <masc-se:MessageToBeSent>
                  <masc-sc:Scope-Message>
                        <masc-sc:ServiceName Name="wsdlfile:WeatherReport"/>
                        <masc-sc:EndpointName Name="wsdlfile:CanadianWeatherReport"/>
                        <masc-sc:OperationName Name="wsdlfile:WeatherTemperature"/>
                        <masc-sc:MessageName Name="wsdlfile:WeatherTemperatureRequest"/>
                  </masc-sc:Scope-Message>
            </masc-se:MessageToBeSent>
      </masc-se:EventDefinition>
      <masc-se:EventDefinition MASCID="ReplyReceived">
            <masc-se:MessageReceipt>
                  <masc-sc:Scope-Message>
                        <masc-sc:ServiceName Name="wsdlfile:WeatherReport"/>
                        <masc-sc:EndpointName Name="wsdlfile:CanadianWeatherReport"/>
                        <masc-sc:OperationName Name="wsdlfile:WeatherTemperature"/>
                        <masc-sc:MessageName Name="wsdlfile:WeatherTemperatureReply"/>
                  </masc-sc:Scope-Message>
            </masc-se:MessageReceipt>
      </masc-se:EventDefinition>
      <masc-se:EventDefinition MASCID="FaultReceived">
            <masc-se:MessageReceipt>
                  <masc-sc:Scope-Message>
                        <masc-sc:ServiceName Name="wsdlfile:WeatherReport"/>
                        <masc-sc:EndpointName Name="wsdlfile:CanadianWeatherReport"/>
                        <masc-sc:OperationName Name="wsdlfile:WeatherTemperature"/>
                        <masc-sc:MessageName Name="wsdlfile:WeatherTemperatureFault"/>
                  </masc-sc:Scope-Message>
            </masc-se:MessageReceipt>
      </masc-se:EventDefinition>
      <!--
                  Definitions of When constructs.
  -->
      <masc-se:When MASCID="Executing-RequestToBeSent">
            <masc-se:AllowedStates>
                  <masc-se:StateRef To="gpExample2:Executing"/>
            </masc-se:AllowedStates>
            <masc-se:PossibleTriggerEvents>
                  <masc-se:EventRef To="tns:RequestToBeSent"/>
            </masc-se:PossibleTriggerEvents>
      </masc-se:When>
      <masc-se:When MASCID="Executing-ReplyOrFaultReceived">
            <masc-se:AllowedStates>
                  <masc-se:StateRef To="gpExample2:Executing"/>
            </masc-se:AllowedStates>
            <masc-se:PossibleTriggerEvents>
                  <masc-se:EventRef To="tns:ReplyReceived"/>
                  <masc-se:EventRef To="tns:FaultReceived"/>
            </masc-se:PossibleTriggerEvents>
      </masc-se:When>
      <!--
                  Definitions of monitoring data collection actions.
  -->
  <masc-ap:MonitoredDataCollection MASCID="MonitoringOfResponseTimeStart-InMilliseconds">
            <masc-gp:MonitoredDataItemRef To="tns:ResponseTimeStart-InMilliseconds"/>
  </masc-ap:MonitoredDataCollection>
```

```
    <masc-ap:MonitoredDataCollection MASCID="MonitoringOfResponseTimeStop-InMilliseconds">
            <masc-gp:MonitoredDataItemRef To="tns:ResponseTimeStop-InMilliseconds"/>
    </masc-ap:MonitoredDataCollection>
    <masc-ap:MonitoredDataCollection MASCID="MonitoringOfResponseTime-InMilliseconds">
            <masc-gp:MonitoredDataItemRef To="tns:ResponseTime-InMilliseconds"/>
            <masc-ap:ConfigurationData>
                    <configData:Start>tns:ResponseTimeStart-InMilliseconds</configData:Start>
                    <configData:Stop>tns:ResponseTimeStop-InMilliseconds</configData:Stop>
            </masc-ap:ConfigurationData>
    </masc-ap:MonitoredDataCollection>
        <!--
                Definitions of monitoring action groups.
    -->
    <masc-ap:ActionGroup MASCID="MonitoringResponseTimeStart">
            <masc-ap:MonitoredDataCollectionRef To="tns:MonitoringOfResponseTimeStart-InMilliseconds"/>
    </masc-ap:ActionGroup>
    <masc-ap:ActionGroup MASCID="MonitoringResponseTimeStop">
            <masc-ap:MonitoredDataCollectionRef To="tns:MonitoringOfResponseTimeStop-InMilliseconds"/>
            <masc-ap:MonitoredDataCollectionRef To="tns:MonitoringOfResponseTime-InMilliseconds"/>
    </masc-ap:ActionGroup>
        <!--
                Definitions of action policy assertions configuring monitoring.
    -->
    <masc-ap:ActionPolicyAssertion MASCID="MonitorResponseTimeStart"
ManagementParty="MASC_WSREQUESTER">
            <masc-se:WhenRef To="tns:Executing-RequestToBeSent"/>
            <masc-ap:ActionGroupRef To="tns:MonitoringResponseTimeStart"/>
    </masc-ap:ActionPolicyAssertion>
    <masc-ap:ActionPolicyAssertion MASCID="MonitorResponseTimeStop"
ManagementParty="MASC_WSREQUESTER">
            <masc-se:WhenRef To="tns:Executing-ReplyReceived"/>
            <masc-ap:ActionGroupRef To="tns:MonitoringResponseTimeStop"/>
    </masc-ap:ActionPolicyAssertion>
        <!--
                Definitions of BooleanExpressions for goal policy assertions.
    -->
        <masc-ex:BooleanExpression MASCID="UpperLimitOfResponseTime">
            <masc-ex:BooleanConstant>true</masc-ex:BooleanConstant>
            <!-- The following elements should be specified instead of the above
            Boolean constraint element that is always true, but this is not yet supported by the
            WS-Policy4MASC grammar (the simplified expressions schema).
            <masc-ex:ArithmeticWitUnitComparator Type="LessOrEqual"/>
            <masc-ex:MonitoredDataItemRef To="tns:ResponseTime-InMilliseconds"/>
            <masc-ex:ArithmeticWitUnitConstant>
                <masc-ex:ArithmeticValue>100</masc-ex:ArithmeticValue>
                <masc-ex:Unit Unit="ontology1:Millisecond"/>
            </masc-ex:ArithmeticWitUnitConstant>
            -->
        </masc-ex:BooleanExpression>
        <!--
                Definitions of goal policy assertions.
    -->
        <masc-gp:GoalPolicyAssertion MASCID="ResponseTimeLimit" ResponsibleParty="MASC_WSPROVIDER"
ManagementParty="MASC_WSPROVIDER">
            <masc-se:WhenRef To="tns:Executing-ReplyReceived"/>
            <masc-ex:BooleanExpressionRef To="tns:UpperLimitOfResponseTime"/>
        </masc-gp:GoalPolicyAssertion>

</wsp:Policy>
```

Appendix 3: The Example File masc-mp-08-Example1.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!-- Example 1 of MetaPolicyAssertion schema (and others) in WS-Policy4MASC version 0.8
        This example is for the Canadian Weather Report example.
        The only Web service operation is: Integer weather(String postalCode).
        The process contains only 1 activity: the invocation of this operation by the orchestrator.
        This example is for the whole process type - the policy must be attached to this process type.
-->
<!--Copyright (c) 2006 The University of New South Wales -->
<!-- NOTE: The following element should also contain the attribute:
wsu:Id="CanadianWhetherReport-WholeOperation-Control"
-->
<wsp:Policy xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
xmlns:masc-gp="http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-gp"
xmlns:masc-se="http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-se"
xmlns:masc-ex="http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-ex" xmlns:masc-
om="http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-om"  xmlns:masc-
sc="http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-sc" xmlns:masc-
ap="http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-ap"
xmlns:masc-up="http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-up"
xmlns:masc-mp="http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-mp"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://schemas.xmlsoap.org/ws/2004/09/policy C:\workspace\WS-Policy4MASCv0-
8\Schemas\ws-policy.xsd http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-gp C:\workspace\WS-
Policy4MASCv0-8\Schemas\ws-policy4masc-gp.xsd
http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-se C:\workspace\WS-Policy4MASCv0-8\Schemas\ws-
policy4masc-se.xsd
http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-ex C:\workspace\WS-Policy4MASCv0-8\Schemas\ws-
policy4masc-ex.xsd
http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-om C:\workspace\WS-Policy4MASCv0-8\Schemas\ws-
policy4masc-sc.xsd
http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-sc C:\workspace\WS-Policy4MASCv0-8\Schemas\ws-
policy4masc-om.xsd
http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-ap C:\workspace\WS-Policy4MASCv0-8\Schemas\ws-
policy4masc-ap.xsd
http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-up C:\workspace\WS-Policy4MASCv0-8\Schemas\ws-
policy4masc-up.xsd
http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-mp C:\workspace\WS-Policy4MASCv0-8\Schemas\ws-
policy4masc-mp.xsd"
targetNamespace="TARGETmasc-mp-Example1" xmlns:tns="masc-mp-Example2"
xmlns:wsdlfile="C:\workspace\WS-Policy4MASCv0-8\Examples\WSDL-Example2.xml"
xmlns:wsdlfile2="C:\workspace\WS-Policy4MASCv0-8\Examples\WSDL-Example2-Replacement.xml"
xmlns:ontology1="C:\workspace\WS-Policy4MASCv0-8\Examples\Ontology-Example2.xml"
xmlns:gpExample2="C:\workspace\WS-Policy4MASCv0-8\Examples\masc-gp-08-Example2-output.xml"
xmlns:gpExample3="C:\workspace\WS-Policy4MASCv0-8\Examples\masc-gp-08-Example3-output.xml">
        <!--
                Definitions of events determining action policy assertions to be executed.
    -->
        <masc-se:EventDefinition MASCID="ResponseTimeLimitNotMet">
                <masc-se:GoalSatisfaction GoalPolicyAssertionID="gpExample3:ResponseTimeLimit"
Satisfied="false"/>
        </masc-se:EventDefinition>
        <!--
                Definitions of When constructs determining action policy assertions to be executed.
    -->
        <masc-se:When MASCID="Executing-ResponseTimeLimitNotMet">
                <masc-se:AllowedStates>
                        <masc-se:StateRef To="gpExample2:Executing"/>
                </masc-se:AllowedStates>
                <masc-se:PossibleTriggerEvents>
```

```
                <masc-se:EventRef To="tns:ResponseTimeLimitNotMet"/>
            </masc-se:PossibleTriggerEvents>
        </masc-se:When>
        <!--
                Definitions of possible actions.
    -->
    <masc-ap:ReplaceBlockWithWSCall-Known MASCID="ProviderWSReplacement"
EndpointURI="http://www.fasterWeatherReport.com/Canada" OperationName="WeatherTemperature">
            <masc-ap:AtProcessPoint After="true"
ReferenceType="MASC_PROCESSRELATIVE">MASC_PROCESSBEGINNING</masc-ap:AtProcessPoint>
            <masc-ap:ToProcessPoint After="false"
ReferenceType="MASC_PROCESSRELATIVE">MASC_PROCESSENDING</masc-ap:ToProcessPoint>
    </masc-ap:ReplaceBlockWithWSCall-Known>
        <!--
                Definitions of possible action groups.
    -->
    <masc-ap:ActionGroup MASCID="DoNothingGroup">
            <!-- This is an empty action group. -->
    </masc-ap:ActionGroup>
    <masc-ap:ActionGroup MASCID="ProviderWSReplacementGroup">
            <masc-ap:MonitoredDataCollectionRef To="tns:ProviderWSReplacement"/>
    </masc-ap:ActionGroup>
        <!--
                Definitions of action policy assertions.
    -->
    <masc-ap:ActionPolicyAssertion MASCID="DoNothingActionPolicyAssertion"
ManagementParty="MASC_WSORCHESTRATOR">
            <masc-se:WhenRef To="tns:Executing-ResponseTimeLimitNotMet"/>
            <masc-ap:ActionGroupRef To="tns:DoNothingGroup"/>
    </masc-ap:ActionPolicyAssertion>
    <masc-ap:ActionPolicyAssertion MASCID="ProviderWSReplacementActionPolicyAssertion"
ManagementParty="MASC_WSORCHESTRATOR">
            <masc-se:WhenRef To="tns:Executing-ResponseTimeLimitNotMet"/>
            <masc-ap:ActionGroupRef To="tns:ProviderWSReplacementGroup"/>
    </masc-ap:ActionPolicyAssertion>
        <!--
                Definitions of events determining utility policy assertions to be calculated.
    -->
        <masc-se:EventDefinition MASCID="DidNothing">
            <masc-se:ActionExecution ActionPolicyAssertionID="tns:DoNothingActionPolicyAssertion"
Completed="true"/>
        </masc-se:EventDefinition>
        <masc-se:EventDefinition MASCID="ProviderWSReplaced">
            <masc-se:ActionExecution
ActionPolicyAssertionID="tns:ProviderWSReplacementActionPolicyAssertion" Completed="true"/>
        </masc-se:EventDefinition>
        <!--
                Definitions of When constructs determining utility policy assertions to be calculated.
    -->
        <masc-se:When MASCID="Executing-DidNothing">
            <masc-se:AllowedStates>
                <masc-se:StateRef To="gpExample2:Executing"/>
            </masc-se:AllowedStates>
            <masc-se:PossibleTriggerEvents>
                <masc-se:EventRef To="tns:DidNothing"/>
            </masc-se:PossibleTriggerEvents>
        </masc-se:When>
        <masc-se:When MASCID="Executing-ProviderWSReplaced">
            <masc-se:AllowedStates>
                <masc-se:StateRef To="gpExample2:Executing"/>
            </masc-se:AllowedStates>
            <masc-se:PossibleTriggerEvents>
                <masc-se:EventRef To="tns:ProviderWSReplaced"/>
```

```
            </masc-se:PossibleTriggerEvents>
        </masc-se:When>
        <!--
                Definitions of arithmetic with unit expressions for utility policy assertions.
    -->
    <masc-ex:ArithmeticWithUnitExpression MASCID="Amount1">
            <masc-ex:ArithmeticWithUnitConstant>
                <masc-ex:ArithmeticValue>-1.00</masc-ex:ArithmeticValue>
                <masc-ex:Unit OntologicalType="ontology1:CanadianDollar"/>
            </masc-ex:ArithmeticWithUnitConstant>
    </masc-ex:ArithmeticWithUnitExpression>
    <masc-ex:ArithmeticWithUnitExpression MASCID="Amount2">
            <masc-ex:ArithmeticWithUnitConstant>
                <masc-ex:ArithmeticValue>10.00</masc-ex:ArithmeticValue>
                <masc-ex:Unit OntologicalType="ontology1:CanadianDollar"/>
            </masc-ex:ArithmeticWithUnitConstant>
    </masc-ex:ArithmeticWithUnitExpression>
    <masc-ex:ArithmeticWithUnitExpression MASCID="Amount3">
            <masc-ex:ArithmeticWithUnitConstant>
                <masc-ex:ArithmeticValue>100.00</masc-ex:ArithmeticValue>
                <masc-ex:Unit OntologicalType="ontology1:CanadianDollar"/>
            </masc-ex:ArithmeticWithUnitConstant>
    </masc-ex:ArithmeticWithUnitExpression>
    <masc-ex:ArithmeticWithUnitExpression MASCID="Amount4">
            <masc-ex:ArithmeticWithUnitConstant>
                <masc-ex:ArithmeticValue>0.00</masc-ex:ArithmeticValue>
                <masc-ex:Unit OntologicalType="ontology1:CanadianDollar"/>
            </masc-ex:ArithmeticWithUnitConstant>
    </masc-ex:ArithmeticWithUnitExpression>
        <!--
                Definitions of utility policy assertions.
    -->
    <masc-up:UtilityPolicyAssertion MASCID="ResponseTimeNotMetPenalty"
ManagementParty="MASC_WSORCHESTRATOR" BeneficiaryParty="MASC_WSPROVIDER"
PayingParty="MASC_WSREQUESTER" IsTangible="true">
            <masc-se:WhenRef To="Executing-ResponseTimeLimitNotMet"/>
            <masc-ex:ArithmeticWithUnitExpressionRef To="tns:Amount1"/>
        </masc-up:UtilityPolicyAssertion>
        <!-- Note that no benefits or costs for Executing-DidNothing are specified. -->
    <masc-up:UtilityPolicyAssertion MASCID="ProviderReplacementWSPrice"
ManagementParty="MASC_WSORCHESTRATOR" BeneficiaryParty="wsdlfile2:FasterWeatherReport"
PayingParty="MASC_WSORCHESTRATOR" IsTangible="true">
            <masc-se:WhenRef To="Executing-ProviderWSReplaced"/>
            <masc-ex:ArithmeticWithUnitExpressionRef To="tns:Amount2"/>
        </masc-up:UtilityPolicyAssertion>
    <masc-up:UtilityPolicyAssertion MASCID="ProviderReplacementWSLongTermBenefit"
ManagementParty="MASC_WSORCHESTRATOR" BeneficiaryParty="MASC_WSORCHESTRATOR"
IsTangible="false">
            <masc-se:WhenRef To="Executing-ProviderWSReplaced"/>
            <masc-ex:ArithmeticWithUnitExpressionRef To="tns:Amount3"/>
        </masc-up:UtilityPolicyAssertion>
        <!--
                Definitions of policy conflict resolution strategies.
        -->
        <masc-mp:PolicyConflictResolutionStrategy MASCID="Strategy-OnlyImmediateAgreedPayments">
            <masc-mp:EstimatedFutureValuesAsTiebreakerOnly>
                <masc-ex:ArithmeticWithUnitExpressionRef To="tns:Amount4"/>
            </masc-mp:EstimatedFutureValuesAsTiebreakerOnly>
        </masc-mp:PolicyConflictResolutionStrategy>
        <masc-mp:PolicyConflictResolutionStrategy MASCID="Strategy-AllImmediatePayments"/>
<!--
                Definitions of various metapolicy assertions.
                NOTE: Only 1 of them can be specified in reality because they are conflicting!
```

```
    -->
        <!-- The following metapolicy assertion specifies that only immediate agreed payments
                (both tangible and intangible) should be considered. Therefore,
                DoNothingActionPolicyAssertion will be chosen. -->
        <masc-mp:MetaPolicyAssertion MASCID="OnlyImmediateAgreedPayments"
ManagementParty="MASC_WSORCHESTRATOR"
PartyForWhichUtilityIsExamined="MASC_WSORCHESTRATOR">
                <masc-ap:ActionPolicyAssertionRef To="DoNothingActionPolicyAssertion"/>
                <masc-ap:ActionPolicyAssertionRef To="ProviderWSReplacementActionPolicyAssertion"/>
                <masc-mp:PolicyConflictResolutionStrategyRef To="Strategy-OnlyImmediateAgreedPayments"/>
        </masc-mp:MetaPolicyAssertion>
        <!-- The following metapolicy assertion specifies that all immediate payments
                (both agreed and estimated, both tangible and intangible) should be considered.
                Therefore, ProviderWSReplacementActionPolicyAssertion will be chosen. -->
        <masc-mp:MetaPolicyAssertion MASCID="AllImmediatePayments"
ManagementParty="MASC_WSORCHESTRATOR"
PartyForWhichUtilityIsExamined="MASC_WSORCHESTRATOR">
                <masc-ap:ActionPolicyAssertionRef To="DoNothingActionPolicyAssertion"/>
                <masc-ap:ActionPolicyAssertionRef To="ProviderWSReplacementActionPolicyAssertion"/>
                <masc-mp:PolicyConflictResolutionStrategyRef To="Strategy-AllImmediatePayments"/>
        </masc-mp:MetaPolicyAssertion>

</wsp:Policy>
```

Appendix 4: The File ws-policy4masc-pa.xsd

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!-- Abstract MASC Policy Assertions Schema of WS-Policy4MASC version 0.8 -->
<!--Copyright (c) 2006 The University of New South Wales -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:tns="http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-pa"
xmlns:masc-pa="http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-pa"
targetNamespace="http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-pa"
elementFormDefault="qualified" attributeFormDefault="unqualified">
 <!--
  MASCConstruct: Definition of an abstract MASC construct type.
 -->
 <xs:complexType name="MASCConstruct">
  <!-- The following attribute defines a unique ID of the policy assertion. -->
  <xs:attribute name="MASCID" type="xs:ID" use="required"/>
 </xs:complexType>
 <!--
 MASCRef: Definition of an abstract reference type to a MASC construct.
 TODO: It might be beneficial to make a hierarchy of MASC references, parallel to the hierarchy of MASC constructs.
 The current solution is simpler to code, but it requires from human WS-Policy4MASC more knowledge and discipline.
 Further, it can report syntactic errors for semantically correct WS-Policy4MASC references when a reference
ELEMENT
 to a supertype is expected and a WS-Policy4MASC file provides a reference ELEMENT to a subtype. However, there
is
 no problem (with the language grammar or language tools) if a reference ELEMENT to a subtype points to an instance
 of a subtype.
 -->
 <xs:complexType name="MASCRef">
  <xs:attribute name="To" type="xs:QName" use="required"/>
 </xs:complexType>
 <!--
 MASCPolicyAssertion (extends MASCConstruct): Definition of an abstract MASC policy asssertion type.
 -->
 <xs:complexType name="MASCPolicyAssertion">
  <xs:complexContent>
   <xs:extension base="masc-pa:MASCConstruct">
    <!-- The following attribute specifies the management party responsible for handling the MASC policy assertion. -->
    <xs:attribute name="ManagementParty" type="xs:anyURI" use="optional"
default="MASC_WSORCHESTRATOR"/>
    <!-- The following attribute specifies whether the MASC policy assertion is initially active or not (for policy
adaptation). -->
    <xs:attribute name="IsActive" type="xs:boolean" use="optional" default="true"/>
    <!-- The following attribute specifies priority - the higher the number, the higher the priority (importance). -->
    <!-- TODO: In the future, priority specification might be done at policy attachment, instead of (or in addition to)
here. -->
    <xs:attribute name="Priority" type="xs:positiveInteger" use="optional" default="1"/>
   </xs:extension>
  </xs:complexContent>
 </xs:complexType>
 <!--
 PolicyAssertionRef (extends MASCRef): Definition of a reference to any goal policy assertion.
 -->
 <xs:element name="MASCPolicyAssertionRef" type="masc-pa:MASCRef"/>
</xs:schema>
```

Appendix 5: The File ws-policy4masc-gp.xsd

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!-- Goal Policy Assertions Schema of WS-Policy4MASC version 0.9 -->
<!--Copyright (c) 2006 The University of New South Wales -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:tns="http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-gp" xmlns:masc-gp="http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-gp" xmlns:masc-pa="http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-pa" xmlns:masc-se="http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-se" xmlns:masc-ex="http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-ex" xmlns:masc-om="http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-om" targetNamespace="http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-gp" elementFormDefault="qualified" attributeFormDefault="unqualified">
	<xs:import namespace="http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-pa" schemaLocation="ws-policy4masc-pa.xsd"/>
	<xs:import namespace="http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-se" schemaLocation="ws-policy4masc-se.xsd"/>
	<xs:import namespace="http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-ex" schemaLocation="ws-policy4masc-ex.xsd"/>
	<xs:import namespace="http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-om" schemaLocation="ws-policy4masc-om.xsd"/>
	<!--
		GoalPolicyAssertion (extends MASCPolicyAssertion): Definition of a goal policy asssertion.
	-->
	<xs:element name="GoalPolicyAssertion" type="masc-gp:GoalPolicyAssertion">
		<xs:annotation>
			<xs:documentation>This element is intended for use as a WS-Policy policy assertion</xs:documentation>
		</xs:annotation>
	</xs:element>
	<xs:complexType name="GoalPolicyAssertion">
		<xs:complexContent>
			<xs:extension base="masc-pa:MASCPolicyAssertion">
				<xs:sequence>
					<!-- The following element is used to reference the precondition for evaluation of the condition to be satisfied. -->
					<xs:element ref="masc-se:WhenRef"/>
					<!-- The following element is used to reference the goal - condition to be satisfied. -->
					<xs:element ref="masc-ex:BooleanExpressionRef"/>
				</xs:sequence>
				<xs:attribute name="ResponsibleParty" type="xs:anyURI" use="optional" default="MASC_WSPROVIDER"/>
			</xs:extension>
			<!-- The ManagementParty attribute (inherited from MASCPolicyAssertion) specifies the management party
				responsible for evaluating this condition -->
		</xs:complexContent>
	</xs:complexType>
	<!--
		GoalPolicyAssertionRef (extends MASCRef): Definition of a reference to a goal policy assertion.
	-->
	<xs:element name="GoalPolicyAssertionRef" type="masc-pa:MASCRef"/>
	<!--
		MonitoredDataItem (extends MASCConstruct): Definition of a monitored (measured or calculated) data item.
	-->
	<xs:element name="MonitoredDataItem" type="masc-gp:MonitoredDataItem">
		<xs:annotation>
			<xs:documentation>This element is intended for use as a WS-Policy policy assertion</xs:documentation>
		</xs:annotation>
```

```
        </xs:element>
        <xs:complexType name="MonitoredDataItem">
                <xs:complexContent>
                        <xs:extension base="masc-pa:MASCConstruct">
                                <xs:sequence>
                                        <!-- At this time, optional ontological meaning is used only for comparisons. In
principle, it can be used as configuration data
                for monitoring/calculation, but this might be too complicated. Therefore, additional element <masc-
ap:ConfigurationData> is
                specified for the element <masc-ap:MonitoredDataItemCollection> in this version of the WS-
Policy4MASC language. -->
                                        <xs:element ref="masc-om:OntologicalMeaningRef" minOccurs="0"/>
                                </xs:sequence>
                                <xs:attribute name="MessagePartName" type="xs:NCName" use="optional"/>
                                <xs:attribute name="ValueDataType" type="xs:QName" use="required"/>
                                <xs:attribute name="Unit" type="xs:QName" use="optional"/>
                        </xs:extension>
                        <!-- If the monitored data item is a message part, then its NCName should be provided here.
                                                (if the monitored data item is not a message part, but a QoS metric, then this
element is not to be specified.) -->
                        <!-- The argument for 'ValueDataType' can be the an xsi:type, but also a complex type. -->
                        <!-- This attribute refers to an ontological defition of the used measurement unit. -->
                </xs:complexContent>
        </xs:complexType>
        <!--
                        MonitoredDataItemRef (extends MASCRef): Definition of a reference to a monitored (measured or
calculated) data item.
    -->
        <xs:element name="MonitoredDataItemRef" type="masc-pa:MASCRef"/>
</xs:schema>
```

Appendix 6: The File ws-policy4masc-ap.xsd

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!-- edited with XMLSpy v2007 sp1 (http://www.altova.com) by Tatjana Tosic (volunteer) -->
<!-- Action Policy Assertions Schema of WS-Policy4MASC version 0.8 -->
<!--Copyright (c) 2006 The University of New South Wales -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:tns="http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-ap" xmlns:masc-ap="http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-ap" xmlns:masc-pa="http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-pa" xmlns:masc-se="http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-se" xmlns:masc-gp="http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-gp" xmlns:masc-ex="http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-ex" targetNamespace="http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-ap" elementFormDefault="qualified" attributeFormDefault="unqualified">
	<xs:import namespace="http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-pa" schemaLocation="ws-policy4masc-pa.xsd"/>
	<xs:import namespace="http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-se" schemaLocation="ws-policy4masc-se.xsd"/>
	<xs:import namespace="http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-gp" schemaLocation="ws-policy4masc-gp.xsd"/>
	<xs:import namespace="http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-ex" schemaLocation="ws-policy4masc-ex.xsd"/>
	<!--
		ActionPolicyAssertion (extends MASCPolicyAssertion): Definition of an action policy asssertion.
	-->
	<xs:element name="ActionPolicyAssertion" type="masc-ap:ActionPolicyAssertion">
		<xs:annotation>
			<xs:documentation>This element is intended for use as a WS-Policy policy assertion</xs:documentation>
		</xs:annotation>
	</xs:element>
	<xs:complexType name="ActionPolicyAssertion">
		<xs:complexContent>
			<xs:extension base="masc-pa:MASCPolicyAssertion">
				<xs:sequence>
					<!-- The following element is used to reference the precondition for execution. -->
					<xs:element ref="masc-se:WhenRef"/>
					<!-- The following element is used to reference the action group to be executed. -->
					<xs:element ref="masc-ap:ActionGroupRef"/>
				</xs:sequence>
			</xs:extension>
			<!-- The ManagementParty attribute (inherited from MASCPolicyAssertion) specifies the
				management party responsible for executing these actions -->
		</xs:complexContent>
	</xs:complexType>
	<!--
		ActionPolicyAssertionRef (extends MASCRef): Definition of a reference to an action policy assertion.
	-->
	<xs:element name="ActionPolicyAssertionRef" type="masc-pa:MASCRef"/>
	<!--
		ActionGroup (extends MASCConstruct): Definition of an action group - a group of actions executed together (at the same time).
		The concept of an action group supports reusability of action descriptions.
	-->
	<xs:element name="ActionGroup" type="masc-ap:ActionGroup">
		<xs:annotation>
			<xs:documentation>This element is intended for use as a WS-Policy policy assertion</xs:documentation>
		</xs:annotation>
	</xs:element>
	<xs:complexType name="ActionGroup">
```

```
            <xs:complexContent>
                <xs:extension base="masc-pa:MASCConstruct">
                    <xs:sequence>
                        <!-- The following lists possible action groups and actions. -->
                        <xs:element ref="masc-ap:ActionGroupRef" minOccurs="0"
maxOccurs="unbounded"/>
                        <xs:element ref="masc-ap:MonitoredDataCollectionRef" minOccurs="0"
maxOccurs="unbounded"/>
                        <xs:element ref="masc-ap:MonitoredDataTransferRef" minOccurs="0"
maxOccurs="unbounded"/>
                        <xs:element ref="masc-ap:ProcessStructureAdaptationRef" minOccurs="0"
maxOccurs="unbounded"/>
                        <xs:element ref="masc-ap:ProcessExecutionAdaptationRef" minOccurs="0"
maxOccurs="unbounded"/>
                        <xs:element ref="masc-ap:ActivityExecutionAdaptationRef" minOccurs="0"
maxOccurs="unbounded"/>
                        <xs:element ref="masc-ap:MessagingAdaptationRef" minOccurs="0"
maxOccurs="unbounded"/>
                        <xs:element ref="masc-ap:PolicyAdaptationRef" minOccurs="0"
maxOccurs="unbounded"/>
                        <!-- Contract adaptation can be added later, when it is finalized whether there will be
a
                             concept of a contract in the WS-Policy4MASC language and how it will be
implemented
                        <xs:element ref="masc-ap:ContractAdaptationRef" minOccurs="0"
maxOccurs="unbounded"/>
                        -->
                        <xs:element ref="masc-ap:ActionCancellationRef" minOccurs="0"
maxOccurs="unbounded"/>
                        <xs:element ref="masc-ap:EventCancellationRef" minOccurs="0"
maxOccurs="unbounded"/>
                    </xs:sequence>
                </xs:extension>
            </xs:complexContent>
        </xs:complexType>
        <!--
                ActionGroupRef (extends MASCRef): Definition of a reference to an action group.
    -->
        <xs:element name="ActionGroupRef" type="masc-pa:MASCRef"/>
        <!--
                !!!
                MonitoredDataCollection (extends MASCConstruct): Definition of a monitoring data collection
(measurement/calculation).
                !!!
    -->
        <xs:element name="MonitoredDataCollection" type="masc-ap:MonitoredDataCollection">
            <xs:annotation>
                <xs:documentation>This element is intended for use as a WS-Policy policy
assertion</xs:documentation>
            </xs:annotation>
        </xs:element>
        <xs:complexType name="MonitoredDataCollection">
            <xs:complexContent>
                <xs:extension base="masc-pa:MASCConstruct">
                    <xs:sequence>
                        <!-- The following element is used to reference the monitored data item to be
collected. -->
                        <xs:element ref="masc-gp:MonitoredDataItemRef"/>
                        <!-- The following element is intended as a placeholder for specifying whatever
configuration data the monitoring party needs.
                             For example, for calculated data items, it could contain an masc-
ex:arithmeticWithUnitExpression describing how to
                             calculate this value. In principle, this configuration data could include the name of
```

```
the implementation module to be used for
                measurement or calculation. However, this brings implementation details into WS-Policy4MASC files, which should be
                avoided. Therefore, it is better to keep internal tables specifying which ontological definitions (of QoS metrics or context
                properties) are monitored by which internal modules. It is likely that there will be a generic MASC middleware module for
                calculations, and probably also generic modules for capturing current time and manipulating counters.This will cover the
                most frequently used QoS metrics. Modules for additional QoS metrics and context properties (e.g., determining
                geographic coordinates using GPS hardware) are outside the scope of this version of the WS-Policy4MASC language and
                the MASC middleware. -->
                <xs:element name="ConfigurationData" minOccurs="0">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
            <xs:attribute name="PassedInSOAPHeaders" type="xs:boolean" use="optional" default="true"/>
        </xs:extension>
        <!-- The following element specifies whether SOAP headers are used as a mechanism for exchange of monitored data items.
                Note that exchange of monitoring information trough pushing/pulling operations (messages) will be defined through action
                policy assertions. -->
    </xs:complexContent>
</xs:complexType>
<!--
        MonitoredDataCollectionRef (extends MASCRef): Definition of a reference to a monitoring data collection (measurement/calculation).
-->
<xs:element name="MonitoredDataCollectionRef" type="masc-pa:MASCRef"/>
<!--
        !!!
        MonitoredDataTransfer (extends MASCConstruct): Definition of a monitoring data transfer.
        !!!
-->
<xs:element name="MonitoredDataTransfer" type="masc-ap:MonitoredDataTransfer">
    <xs:annotation>
        <xs:documentation>This element is intended for use as a WS-Policy policy assertion</xs:documentation>
    </xs:annotation>
</xs:element>
<xs:complexType name="MonitoredDataTransfer">
    <xs:complexContent>
        <xs:extension base="masc-pa:MASCConstruct">
            <xs:sequence>
                <!-- The following element is used to reference the monitored data items to be transferred. -->
                <xs:element ref="masc-gp:MonitoredDataItemRef" maxOccurs="unbounded"/>
            </xs:sequence>
            <xs:attribute name="TransferPartner" type="xs:anyURI" use="optional" default="MASC_WSORCHESTRATOR"/>
            <xs:attribute name="TransferType" type="xs:string" use="optional" default="push"/>
        </xs:extension>
        <!-- At this time, 'TransferType' can be either "push" or "pull".
                TODO: For now, it is assumed that there are standard MASC interfaces for pulling
```

```
and pushing. Later, description of
                specific Web services, endpoints, operations, and parameters for pulling/pushing could be added. --
>
                <!-- This is the QName of the other transfer party, in addition to the 'ManagementParty' that
executes the action policy.
                        That is, for "push" transfer, 'ManagementParty' is data source and 'TransferPartner'
is data recipient,
                        while for "pull" transfer, 'ManagementParty' is data recipient and 'TransferParty' is
data source.
                -->
            </xs:complexContent>
        </xs:complexType>
        <!--
                MonitoredDataTransferRef (extends MASCRef): Definition of a reference to a monitoring data
transfer.
    -->
        <xs:element name="MonitoredDataTransferRef" type="masc-pa:MASCRef"/>
        <!--
                !!!
                ProcessStructureAdaptation (extends MASCConstruct): Definition of an abstract process structure
adaptation type.
                !!!
    -->
        <xs:complexType name="ProcessStructureAdaptation">
            <xs:complexContent>
                <xs:extension base="masc-pa:MASCConstruct">
                    <xs:sequence>
                        <!-- The following element is used to specify the process point where the process
structure adaptation is performed. -->
                        <xs:element ref="masc-ap:AtProcessPoint"/>
                        <!-- The following element is used to specify process adaptation data transfers (e.g.,
variable bindings)
                                between the the base process and the external added process (and/or vice
versa), if any. -->
                        <xs:element ref="masc-ap:ProcessAdaptationDataTransfer" minOccurs="0"
maxOccurs="unbounded"/>
                    </xs:sequence>
                    <xs:attribute name="ForThisIterationOnly" type="xs:boolean" use="optional"
default="false"/>
                </xs:extension>
                <!-- This attribute specifies whether the change is performed only for the current iteration (if the
adapted process instance
                        is executing and is currently in a loop) or for all iterations. The defualt case is that
the change is performed for all
                        iterations and it is used for both static adaptation and dynamic adaptation. In MASC,
this case is implemented with
                        modifications to the process instance definition. On the other hand, if the value of
this attribute is "true", then the change
                        is performed only for the current iteration, which is applicable only to dynamic
adaptation. In MASC, this case is
                        implemented with manipulation of the execution queues.
                    -->
            </xs:complexContent>
        </xs:complexType>
        <!--
                ProcessStructureAdaptationRef (extends MASCRef): Definition of a reference to any process
adaptation.
    -->
        <xs:element name="ProcessStructureAdaptationRef" type="masc-pa:MASCRef"/>
        <!--
                AtProcessPoint: Definition of the process point at (or from) which something is done.
                ToProcessPoint: Definition of the process point to which something is done.
    -->
```

```
        <xs:element name="AtProcessPoint" type="masc-ap:ProcessPoint"/>
        <xs:element name="ToProcessPoint" type="masc-ap:ProcessPoint"/>
        <!--
                        ProcessPoint: Definition of the process point type.
    -->
        <xs:complexType name="ProcessPoint">
                <xs:simpleContent>
                        <xs:extension base="xs:string">
                                <xs:attribute name="After" type="xs:boolean" default="true"/>
                                <xs:attribute name="ReferenceType" type="xs:string" default="XPath"/>
                        </xs:extension>
                </xs:simpleContent>
                <!-- The following element is used to specify whether the specified point is included or not.
                        This can be useful to specify '(' or ')' type of limits of intervals instead of '[' or ']' type of limits.
                        Further, it can be used to specify 'before' or 'after' semantics associated to a process point.
                        For example, if the specified process point is used for addition into the base process and
After="true" then
                        the addition performed "after" this point. On the other hand, if the specified process point is used
for addition into the base
                        process and After="false" then the addition performed "before this point.
                -->
                <!-- The following element is used to specify what type of format is used to specify the
                        content of an element of the encompassing complexType type. -->
        </xs:complexType>
        <!--
                        ProcessAdaptationDataTransfer: Definition of a process adaptation data transfer (e.g., variable
binding)
                                                                                between the the base
process and the the external added process (and/or vice versa).
                        TODO: At this time, only straightforward mapping of variables in the base process and the added
process is supported.
                                        This means that data transformation has to be modeled through additional process
structure additions.
                                        In the future, a more powerful support for data transformation through WS-
Policy4MASC expressions will be provided.
    -->
        <xs:element name="ProcessAdaptationDataTransfer" type="masc-ap:ProcessAdaptationDataTransfer"/>
        <xs:complexType name="ProcessAdaptationDataTransfer">
                <xs:sequence>
                        <!-- The following element specifies the variable in the base process. -->
                        <xs:element name="BaseProcessVariableRef" type="masc-ex:ProcessVariableRef"/>
                        <!-- The following element specifies the variable in the the external added process. -->
                        <xs:element name="AddedProcessVariableRef" type="masc-ex:ProcessVariableRef"/>
                        <!-- TODO:  This might be one possible way to specify data transformation (but it is not yet
tested). -->
                        <xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
                </xs:sequence>
                <xs:attribute name="PerformedOn" use="required">
                        <xs:simpleType>
                                <xs:restriction base="xs:string">
                                        <xs:enumeration value="CallToProcessVariationOnly"/>
                                        <xs:enumeration value="ReturnFromProcessVariationOnly"/>
                                        <xs:enumeration value="BothCallToAndReturnFromProcessVariation"/>
                                        <xs:enumeration value="Always-BoundVariables"/>
                                        <!-- Copy from the base process to the added process. -->
                                        <!-- Copy from the added process to the base process. -->
                                        <!-- Copy both from the base process to the added process and the other way around.
-->
                                        <!-- Define a variable binding. -->
                                </xs:restriction>
                        </xs:simpleType>
                </xs:attribute>
                <!-- The following attribute describes when the process adaptation data transfer is performed. -->
```

```
            <!-- The following attribute specifies when and how to perform the data transfer. -->
        </xs:complexType>
        <!--
                RemoveBaseProcessBlock (extends ProcessStructureAdaptation): Definition of a removal of a
process blok

                                                                from the base process.
    -->
        <xs:element name="RemoveBaseProcessBlock" type="masc-ap:RemoveBaseProcessBlock">
            <xs:annotation>
                <xs:documentation>This element is intended for use as a WS-Policy policy
assertion</xs:documentation>
            </xs:annotation>
        </xs:element>
        <xs:complexType name="RemoveBaseProcessBlock">
            <xs:complexContent>
                <xs:extension base="masc-ap:ProcessStructureAdaptation">
                    <xs:sequence>
                        <!-- The following element is used to specify the process point to which the removal
is performed.
                                (Note that the removal is performed from the point specified in the inherited
'AtProcessPoint' element.) -->
                        <xs:element ref="masc-ap:ToProcessPoint"/>
                    </xs:sequence>
                </xs:extension>
            </xs:complexContent>
        </xs:complexType>
        <!--
                RemoveBaseProcessBlockRef (extends MASCRef): Definition of a reference to a removal of a
process blok from the base process.
    -->
        <xs:element name="RemoveBaseProcessBlockRef" type="masc-pa:MASCRef"/>
        <!--
                RemoveBaseProcessActivity (extends ProcessStructureAdaptation): It might be useful to have a
separate element for removal of

                                                        only one activity from the base process.
                                                                However, this is a special
case of RemoveBaseProcessBlock, with which it can be modeled easily
                                                                (although somewhat
verbosely). Therefore, RemoveBaseProcessActivity is not included, for now.
    -->
        <!--
                AddExternalProcess-Known (extends ProcessStructureAdaptation): Definition of an addition of a
known external process

                                                                as a variation to the base process.
    -->
        <xs:element name="AddExternalProcess-Known" type="masc-ap:AddExternalProcess-Known">
            <xs:annotation>
                <xs:documentation>This element is intended for use as a WS-Policy policy
assertion</xs:documentation>
            </xs:annotation>
        </xs:element>
        <xs:complexType name="AddExternalProcess-Known">
            <xs:complexContent>
                <xs:extension base="masc-ap:ProcessStructureAdaptation">
                    <xs:attribute name="AddedProcessQName" type="xs:QName" use="required"/>
                </xs:extension>
                <!-- The following attribute specifies the qualified name of the process (or the called Web service)
to be added. -->
            </xs:complexContent>
        </xs:complexType>
```

```
        <!--
                AddExternalProcess-KnownRef (extends MASCRef): Definition of a reference to an addition of a known external process

                        as a variation to the base process.
    -->
        <xs:element name="AddExternalProcess-KnownRef" type="masc-pa:MASCRef"/>
        <!--
                AddWSCall-Known (extends ProcessStructureAdaptation): Definition of an addition of one activity that is a call to a

                                known external Web service as a variation to the base

                                process. (The activity should be generated automatically.)
    -->
        <xs:element name="AddWSCall-Known" type="masc-ap:AddWSCall-Known">
                <xs:annotation>
                        <xs:documentation>This element is intended for use as a WS-Policy policy assertion</xs:documentation>
                </xs:annotation>
        </xs:element>
        <xs:complexType name="AddWSCall-Known">
                <xs:complexContent>
                        <xs:extension base="masc-ap:ProcessStructureAdaptation">
                                <xs:attribute name="EndpointURI" type="xs:anyURI" use="required"/>
                                <xs:attribute name="OperationName" type="xs:NCName" use="required"/>
                        </xs:extension>
                        <!-- The following attribute specifies the URI of the Web service endpoint (port) called. -->
                        <!-- The following attribute specifies the NCName of the Web service operation called. -->
                </xs:complexContent>
        </xs:complexType>
        <!--
                AddWSCall-KnownRef (extends MASCRef): Definition of a reference to an addition of one activity that is a call to a

                known external Web service as a variation to the base process.
    -->
        <xs:element name="AddWSCall-KnownRef" type="masc-pa:MASCRef"/>
        <!--
                AddExternalProcessOrWSCall-Searched (extends ProcessStructureAdaptation): Definition of an addition of an external process

                                or a Web service call (to be found) as a variation to the base process.
    -->
        <!--
                NOTE: Further constructs for addition of external processes or Web service operation calls can be added, such as addition at
                        multiple places, addition around a process block, etc. However, they can be easily (although somewhat verbosely)
                        modeled with multiple calls to the current addition constructs.
    -->
        <xs:element name="AddExternalProcessOrWSCall-Searched" type="masc-ap:AddExternalProcessOrWSCall-Searched">
                <xs:annotation>
                        <xs:documentation>This element is intended for use as a WS-Policy policy assertion</xs:documentation>
                </xs:annotation>
        </xs:element>
        <xs:complexType name="AddExternalProcessOrWSCall-Searched">
                <xs:complexContent>
                        <xs:extension base="masc-ap:ProcessStructureAdaptation">
                                <xs:sequence>
                                        <!-- The following element specifies search and selection parameters, if any. -->
```

```
                        <xs:element ref="masc-ap:SearchAndSelectionParameter" minOccurs="0"
maxOccurs="unbounded"/>
                    </xs:sequence>
                    <xs:attribute name="SearchAndSelectionURI" type="xs:anyURI" use="required"/>
                    <xs:attribute name="ConstructSearchFor" use="required">
                        <xs:simpleType>
                            <xs:restriction base="xs:string">
                                <xs:enumeration value="Process"/>
                                <xs:enumeration value="WebServiceOperation"/>
                                <!-- Search for a process and add it. -->
                                <!-- Search for a Web service operation and add an (automatically
generated) activity that calls it. -->
                            </xs:restriction>
                        </xs:simpleType>
                    </xs:attribute>
                </xs:extension>
                <!-- The following attribute specifies the URI where to search for the process. -->
                <!-- The following attribute specifies the type of the searched construct. -->
            </xs:complexContent>
        </xs:complexType>
        <!--
                AddExternalProcessOrWSCall-SearchedRef (extends MASCRef): Definition of a reference to an
addition of an external process
                                or a Web service call (to be found) as a variation to the base process.
    -->
        <xs:element name="AddExternalProcessOrWSCall-SearchedRef" type="masc-pa:MASCRef"/>
        <!--
                SearchAndSelectionParameter: Definition of a parameter for process or Web service search and
selection.
                TODO: At this time, only constant values of search and selection parameters are supported.
Enabling that any
                        expression can be used for a parameter value is left for future work.
    -->
        <xs:element name="SearchAndSelectionParameter" type="masc-ap:SearchAndSelectionParameter"/>
        <xs:complexType name="SearchAndSelectionParameter">
            <xs:sequence>
                <!-- TODO:  This might be one possible way to specify complex assignment of parameter values
(but it is not yet tested). -->
                <xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
            </xs:sequence>
            <xs:attribute name="ParameterName" type="xs:string" use="required"/>
            <xs:attribute name="ParameterValueDataType" type="xs:QName" use="required"/>
            <xs:attribute name="ParameterValue" type="xs:string" use="required"/>
            <!-- Name of the parameter. -->
            <!-- Data type of the parameter. (Required because the actual value can be of any data type and is written
as a string in
                 WS-Policy4MASC files, so casting is necessary.) -->
            <!-- Constant value of the parameter (specified as a string). -->
        </xs:complexType>
        <!--
                ForkExternalProcess-Known, ForkWSCall-Known, ForkExternalProcessOrWSCall-Searched:
These constructs would
                specify that a known external process, a known external Web service operation, or a found
(searched and selected)
                external process or Web service operation are added in parallel with the regular flow of the
base process.
                TODO: While these constructs might be useful in some situations, their implementation in
the MASC middleware is
                        somewhat complicated, so they are left for possible future work (but with
low priority).
    -->
        <!--
```

```
                NOTE: Further constructs for addition of external processes or Web service operation calls can be added, such as addition at
                multiple places, addition around a process block, etc. However, they can be easily (although somewhat verbosely)
                modeled with multiple calls to the current addition constructs.
    -->
    <!--
                NOTE: While replacement could be relatively easily modeled as a removal + an addition, we decided to model it explicitly
                because it is a common case and the meaning of replacement is hidden when it is modeled in the above way.
    -->
    <!--
                ReplaceBlockWithExternalProcess-Known (extends AddExternalProcess-Known):
                    Definition of a replacement of process block in the base process with a known external process.
    -->
    <xs:element name="ReplaceBlockWithExternalProcess-Known" type="masc-ap:ReplaceBlockWithExternalProcess-Known">
        <xs:annotation>
            <xs:documentation>This element is intended for use as a WS-Policy policy assertion</xs:documentation>
        </xs:annotation>
    </xs:element>
    <xs:complexType name="ReplaceBlockWithExternalProcess-Known">
        <xs:complexContent>
            <xs:extension base="masc-ap:AddExternalProcess-Known">
                <xs:sequence>
                    <!-- The following element is used to specify the process point to which the replacement is performed.
                        (Note that the replacement is performed from the point specified in the inherited 'AtProcessPoint' element.) -->
                    <xs:element ref="masc-ap:ToProcessPoint"/>
                </xs:sequence>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
    <!--
                ReplaceBlockWithExternalProcess-KnownRef (extends MASCRef): Definition of a reference to replacement of process block in the
                base process with a known external process.
    -->
    <xs:element name="ReplaceBlockWithExternalProcess-KnownRef" type="masc-pa:MASCRef"/>
    <!--
                ReplaceBlockWithWSCall-Known (extends AddWSCall-Known):
                    Definition of a replacement of process block in the base process with a call to a known Web service.
                    (The activity should be generated automatically.)
    -->
    <xs:element name="ReplaceBlockWithWSCall-Known" type="masc-ap:ReplaceBlockWithWSCall-Known">
        <xs:annotation>
            <xs:documentation>This element is intended for use as a WS-Policy policy assertion</xs:documentation>
        </xs:annotation>
    </xs:element>
    <xs:complexType name="ReplaceBlockWithWSCall-Known">
        <xs:complexContent>
            <xs:extension base="masc-ap:AddWSCall-Known">
                <xs:sequence>
                    <!-- The following element is used to specify the process point to which the replacement is performed.
                        (Note that the replacement is performed from the point specified in the
```

```
inherited 'AtProcessPoint' element.) -->
                                <xs:element ref="masc-ap:ToProcessPoint"/>
                            </xs:sequence>
                        </xs:extension>
                    </xs:complexContent>
                </xs:complexType>
                <!--
                        ReplaceBlockWithWSCall-KnownRef (extends MASCRef): Definition of a reference to a
replacement of process block in the base

                                process with a call to a known Web service.
    -->
                <xs:element name="ReplaceBlockWithWSCall-KnownRef" type="masc-pa:MASCRef"/>
                <!--
                        ReplaceBlockWithExternalProcessOrWSCall-Searched (extends AddExternalProcessOrWSCall-
Searched):
                            Definition of a replacement of process block in the base process with a call to a known Web
service.
                            (The activity should be generated automatically.)
    -->
                <xs:element name="ReplaceBlockWithExternalProcessOrWSCall-Searched" type="masc-
ap:ReplaceBlockWithExternalProcessOrWSCall-Searched">
                    <xs:annotation>
                        <xs:documentation>This element is intended for use as a WS-Policy policy
assertion</xs:documentation>
                    </xs:annotation>
                </xs:element>
                <xs:complexType name="ReplaceBlockWithExternalProcessOrWSCall-Searched">
                    <xs:complexContent>
                        <xs:extension base="masc-ap:AddExternalProcessOrWSCall-Searched">
                            <xs:sequence>
                                <!-- The following element is used to specify the process point to which the
replacement is performed.
                                        (Note that the replacement is performed from the point specified in the
inherited 'AtProcessPoint' element.) -->
                                <xs:element ref="masc-ap:ToProcessPoint"/>
                            </xs:sequence>
                        </xs:extension>
                    </xs:complexContent>
                </xs:complexType>
                <!--
                        ReplaceBlockWithExternalProcessOrWSCall-SearchedRef (extends MASCRef):
                            Definition of a reference to a .
    -->
                <xs:element name="ReplaceBlockWithExternalProcessOrWSCall-SearchedRef" type="masc-pa:MASCRef"/>
                <!--
                        !!!
                        ProcessExecutionAdaptation (extends MASCConstruct): Definition of an abstract process
execution adaptation type.
                        !!!
    -->
                <xs:complexType name="ProcessExecutionAdaptation">
                    <xs:complexContent>
                        <xs:extension base="masc-pa:MASCConstruct">
                            <xs:attribute name="ProcessInstanceID" type="xs:string"
default="MASC_EVENTPROCESSINSTANCE"/>
                        </xs:extension>
                        <!-- TODO: At this time, there is no element or attribute here. In the future, some attributes (e.g.,
about where to log
                                                and/or who to notify) might be added. -->
                        <!-- This attribute specifies the process instance that is adapted. -->
                    </xs:complexContent>
                </xs:complexType>
```

```
        <!--
                ProcessExecutionAdaptationRef (extends MASCRef): Definition of a reference to any process
execution adaptation.
    -->
        <xs:element name="ProcessExecutionAdaptationRef" type="masc-pa:MASCRef"/>
        <!--
                TerminateProcess (extends ProcessExecutionAdaptation): Definition of a process termination.
    -->
        <xs:element name="TerminateProcess" type="masc-ap:TerminateProcess">
            <xs:annotation>
                <xs:documentation>This element is intended for use as a WS-Policy policy
assertion</xs:documentation>
            </xs:annotation>
        </xs:element>
        <xs:complexType name="TerminateProcess">
            <xs:complexContent>
                <xs:extension base="masc-ap:ProcessExecutionAdaptation"/>
                <!-- TODO: At this time, there is no element or attribute here. In the future, some attributes (e.g.,
about where to log
                                        and/or who to notify) might be added. -->
            </xs:complexContent>
        </xs:complexType>
        <!--
                TerminateProcessRef (extends MASCRef): Definition of a reference to process termination.
    -->
        <xs:element name="TerminateProcessRef" type="masc-pa:MASCRef"/>
        <!--
                SuspendProcess (extends ProcessExecutionAdaptation): Definition of a process suspension

                                        (until a separately defined resumption).
    -->
        <xs:element name="SuspendProcess" type="masc-ap:SuspendProcess">
            <xs:annotation>
                <xs:documentation>This element is intended for use as a WS-Policy policy
assertion</xs:documentation>
            </xs:annotation>
        </xs:element>
        <xs:complexType name="SuspendProcess">
            <xs:complexContent>
                <xs:extension base="masc-ap:ProcessExecutionAdaptation"/>
                <!-- TODO: At this time, there is no element or attribute here. In the future, some attributes (e.g.,
about where to log
                                        and/or who to notify) might be added. -->
            </xs:complexContent>
        </xs:complexType>
        <!--
                SuspendProcessRef (extends MASCRef): Definition of a reference to process suspension

         (until a separately defined resumption).
    -->
        <xs:element name="SuspendProcessRef" type="masc-pa:MASCRef"/>
        <!--
                ResumeProcess (extends ProcessExecutionAdaptation): Definition of a process resumption (if it
was suspended previously).
    -->
        <xs:element name="ResumeProcess" type="masc-ap:ResumeProcess">
            <xs:annotation>
                <xs:documentation>This element is intended for use as a WS-Policy policy
assertion</xs:documentation>
            </xs:annotation>
        </xs:element>
        <xs:complexType name="ResumeProcess">
            <xs:complexContent>
```

```
                    <xs:extension base="masc-ap:ProcessExecutionAdaptation"/>
                    <!-- TODO: At this time, there is no element or attribute here. In the future, some attributes (e.g., about where to log
                                and/or who to notify) might be added. -->
                </xs:complexContent>
            </xs:complexType>
            <!--
                    ResumeProcessRef (extends MASCRef): Definition of a reference to a process resumption (if it was suspended previously).
        -->
            <xs:element name="ResumeProcessRef" type="masc-pa:MASCRef"/>
            <!--
                    !!!
                    ActivityExecutionAdaptation (extends MASCConstruct): Definition of an abstract activity execution adaptation type.
                    NOTE: For activity execution adaptation, it assumed that the adaptation is performed at (or next to) the current execution
                            point of the process instance, so there is usually no need for an explict 'AtProcessPoint' element.
                            (If the process instance has not yet started, then the process beginning is assumed as the current point.
                            If the process instance already finished execution, but still exists, then the process ending is assumed.)
                    NOTE: However, it is an issue what is the "current execution point" when multiple parallel threads (forks) are running. Therefore, we have introduced the ""ProcessInstanceThreadID" attribute to capture thread ID.
                    NOTE: An alternative would be that action policies containing activity execution adaptations can be attached only to particular
                            activities, not a process type (or a process instance). This removes the problem noted above, but requires that such
                            action policies are defined separately (to prevent semantic errors). Also, it is not clear what the meaning of performing
                            these actions (e.g., compensation) on an activity that has not yet executed is. Therefore, we have adopted that action
                            policies containing activity execution adaptations are attached to a process type (or a process instance) and apply only
                            to a current (or near by) activity. If such an action has to be used for another activity, then a corresponding
                            ProcessStructureAdaptation with the attribute ForThisIterationOnly="true" must be used.
                    !!!
        -->
            <xs:complexType name="ActivityExecutionAdaptation">
                <xs:complexContent>
                    <xs:extension base="masc-ap:ProcessExecutionAdaptation">
                        <xs:sequence>
                            <!-- The following element is used to specify process adaptation data transfers (e.g., variable bindings)
                                        between the the base process and the external added process (and/or vice versa), if any. -->
                            <xs:element ref="masc-ap:ProcessAdaptationDataTransfer" minOccurs="0" maxOccurs="unbounded"/>
                        </xs:sequence>
                        <xs:attribute name="ProcessInstanceThreadID" type="xs:string" default="MASC_EVENTTHREAD"/>
                    </xs:extension>
                    <!-- This attribute specifies the process instance thread/fork that is adapted. -->
                </xs:complexContent>
            </xs:complexType>
            <!--
                    ActivityExecutionAdaptationRef (extends MASCRef): Definition of a reference to any process execution adaptation.
        -->
            <xs:element name="ActivityExecutionAdaptationRef" type="masc-pa:MASCRef"/>
```

```
        <!--
                CancelCurrentActivity (extends ActivityExecutionAdaptation): Definition of a cancellation of a
current activity.
                                                                NOTE: For an activity that is not current,
use RemoveBaseProcessBlock.
    -->
        <xs:element name="CancelCurrentActivity" type="masc-ap:CancelCurrentActivity">
            <xs:annotation>
                <xs:documentation>This element is intended for use as a WS-Policy policy
assertion</xs:documentation>
            </xs:annotation>
        </xs:element>
        <xs:complexType name="CancelCurrentActivity">
            <xs:complexContent>
                <xs:extension base="masc-ap:ActivityExecutionAdaptation"/>
                <!-- TODO: At this time, there is no element or attribute here. In the future, some attributes (e.g.,
about where to log
                                                and/or who to notify) might be added. -->
            </xs:complexContent>
        </xs:complexType>
        <!--
                CancelCurrentActivityRef (extends MASCRef): Definition of a reference to a cancellation of a
current activity.
    -->
        <xs:element name="CancelCurrentActivityRef" type="masc-pa:MASCRef"/>
        <!--
                SkipNextActivity (extends ActivityExecutionAdaptation): Definition of a skipping of a next
activity.
                                                                NOTE: For an activity that is not current,
use RemoveBaseProcessBlock.
    -->
        <xs:element name="SkipNextActivity" type="masc-ap:SkipNextActivity">
            <xs:annotation>
                <xs:documentation>This element is intended for use as a WS-Policy policy
assertion</xs:documentation>
            </xs:annotation>
        </xs:element>
        <xs:complexType name="SkipNextActivity">
            <xs:complexContent>
                <xs:extension base="masc-ap:ActivityExecutionAdaptation"/>
                <!-- TODO: At this time, there is no element or attribute here. In the future, some attributes (e.g.,
about where to log
                                                and/or who to notify) might be added. -->
            </xs:complexContent>
        </xs:complexType>
        <!--
                SkipNextActivityRef (extends MASCRef): Definition of a reference to a skipping of a next
activity.
    -->
        <xs:element name="SkipNextActivityRef" type="masc-pa:MASCRef"/>
        <!--
                SkipBlockFromNextActivity (extends SkipNextActivity): Definition of a skipping of a process
block from the next activity to

                                                the given activity.
                                                                NOTE: For an activity that is not current,
use RemoveBaseProcessBlock.
    -->
        <xs:element name="SkipBlockFromNextActivity" type="masc-ap:SkipBlockFromNextActivity">
            <xs:annotation>
                <xs:documentation>This element is intended for use as a WS-Policy policy
assertion</xs:documentation>
            </xs:annotation>
```

```
        </xs:element>
        <xs:complexType name="SkipBlockFromNextActivity">
            <xs:complexContent>
                <xs:extension base="masc-ap:SkipNextActivity">
                    <xs:sequence>
                        <!-- The following element is used to specify the process point up to which the
skipping is performed. -->
                        <xs:element ref="masc-ap:ToProcessPoint"/>
                    </xs:sequence>
                </xs:extension>
            </xs:complexContent>
        </xs:complexType>
        <!--
                SkipBlockFromNextActivityRef (extends MASCRef): Definition of a reference to a skipping of a a
process block from

                the next activity to the given activity.
    -->
        <xs:element name="SkipBlockFromNextActivityRef" type="masc-pa:MASCRef"/>
        <!--
                RescheduleNextActivity (extends SkipNextActivity): Definition of a rescheduling of the next
activity for a later point in the process.
                NOTE: For an activity that is not current,
use RemoveBaseProcessBlock plus AddExternalProcess-Known.
    -->
        <xs:element name="RescheduleNextActivity" type="masc-ap:RescheduleNextActivity">
            <xs:annotation>
                <xs:documentation>This element is intended for use as a WS-Policy policy
assertion</xs:documentation>
            </xs:annotation>
        </xs:element>
        <xs:complexType name="RescheduleNextActivity">
            <xs:complexContent>
                <xs:extension base="masc-ap:SkipNextActivity">
                    <xs:sequence>
                        <!-- The following element is used to specify the process point to which the
rescheduling is performed. -->
                        <xs:element ref="masc-ap:ToProcessPoint"/>
                    </xs:sequence>
                </xs:extension>
            </xs:complexContent>
        </xs:complexType>
        <!--
                RescheduleNextActivityRef (extends MASCRef): Definition of a reference to a rescheduling of the
next activity for a later point

                  in the process.
    -->
        <xs:element name="RescheduleNextActivityRef" type="masc-pa:MASCRef"/>
        <!--
                CompensateLastActivity (extends ActivityExecutionAdaptation): Definition of a compensation of
the last successfully completed

                activity. It is assumed that the implementation of this
activity has

                a predefined (and built-in) compensation operation.
                NOTE: For an activity that is not last one or does not have a predefiined compensation operation,
use AddExternalProcess-Known
                or AddWSCall-Known (to call a compensation operation).
    -->
        <xs:element name="CompensateLastActivity" type="masc-ap:CompensateLastActivity">
            <xs:annotation>
```

```
                <xs:documentation>This element is intended for use as a WS-Policy policy
assertion</xs:documentation>
            </xs:annotation>
        </xs:element>
        <xs:complexType name="CompensateLastActivity">
            <xs:complexContent>
                <xs:extension base="masc-ap:ActivityExecutionAdaptation"/>
                <!-- TODO: At this time, there is no element or attribute here. In the future, some attributes (e.g.,
about where to log
                                and/or who to notify) might be added. -->
            </xs:complexContent>
        </xs:complexType>
        <!--
                CompensateLastActivityRef (extends MASCRef): Definition of a reference to a compensation of
the last successfully

                        completed activity.
    -->
        <xs:element name="CompensateLastActivityRef" type="masc-pa:MASCRef"/>
        <!--
                RetryLastActivity (extends ActivityExecutionAdaptation): Definition of a retrial of the last activity
(that was not completed).
    -->
        <xs:element name="RetryLastActivity" type="masc-ap:RetryLastActivity">
            <xs:annotation>
                <xs:documentation>This element is intended for use as a WS-Policy policy
assertion</xs:documentation>
            </xs:annotation>
        </xs:element>
        <xs:complexType name="RetryLastActivity">
            <xs:complexContent>
                <xs:extension base="masc-ap:ActivityExecutionAdaptation">
                    <xs:sequence>
                        <!-- This element specifies the interval between schedued retries (but the first retry is
performed as soon as possible). -->
                        <xs:element ref="masc-se:IntervalDuration"/>
                        <!-- This element specifies the maximum number of repetitions -->
                        <xs:element ref="masc-se:MaxRepetitions"/>
                    </xs:sequence>
                </xs:extension>
            </xs:complexContent>
        </xs:complexType>
        <!--
                RetryLastActivityRef (extends MASCRef): Definition of a reference to a retrial of the last activity
(that was not completed).
    -->
        <xs:element name="RetryLastActivityRef" type="masc-pa:MASCRef"/>
        <!--
                SuspendCurrentActivity (extends ActivityExecutionAdaptation): Definition of a suspension of the
current activity

                                (until a separately defined resumption).
                                        NOTE: The difference between suspension of an activity
and a whole process is that if a process has
                                                more than one parallel thread, suspending
an activity suspends only its thread and not the other threads.
    -->
        <xs:element name="SuspendCurrentActivity" type="masc-ap:SuspendCurrentActivity">
            <xs:annotation>
                <xs:documentation>This element is intended for use as a WS-Policy policy
assertion</xs:documentation>
            </xs:annotation>
        </xs:element>
```

```
        <xs:complexType name="SuspendCurrentActivity">
            <xs:complexContent>
                <xs:extension base="masc-ap:ActivityExecutionAdaptation"/>
                <!-- TODO: At this time, there is no element or attribute here. In the future, some attributes (e.g., about where to log
                                                and/or who to notify) might be added. -->
            </xs:complexContent>
        </xs:complexType>
        <!--
                SuspendCurrentActivityRef (extends MASCRef): Definition of a reference to a suspension of the current activity

         (until a separately defined resumption).
    -->
        <xs:element name="SuspendCurrentActivityRef" type="masc-pa:MASCRef"/>
        <!--
                ResumeActivity (extends ActivityExecutionAdaptation): Definition of an activity resumption (if it was suspended previously).
    -->
        <xs:element name="ResumeActivity" type="masc-ap:ResumeActivity">
            <xs:annotation>
                <xs:documentation>This element is intended for use as a WS-Policy policy assertion</xs:documentation>
            </xs:annotation>
        </xs:element>
        <xs:complexType name="ResumeActivity">
            <xs:complexContent>
                <xs:extension base="masc-ap:ActivityExecutionAdaptation">
                    <xs:sequence>
                        <!-- This element points the resumed activity, because several activities
                                                can be suspended at the same time. -->
                        <xs:element name="AtProcessPoint" type="masc-ap:ProcessPoint"/>
                        <!-- TODO: At this time, there is no other element or attribute here. In the future, some attributes (e.g., about where to log
                                                and/or who to notify) might be added. -->
                    </xs:sequence>
                </xs:extension>
            </xs:complexContent>
        </xs:complexType>
        <!--
                ResumeActivityRef (extends MASCRef): Definition of a reference to an activity resumption (if it was suspended previously).
    -->
        <xs:element name="ResumeActivityRef" type="masc-pa:MASCRef"/>
        <!--    ChangeActivityParameters:
                TODO: This construct was not defined, because it is not clear what are the changed parameters and where they are defined.
        -->
        <!--
                !!!
                MessagingAdaptation (extends MASCConstruct): Definition of an abstract messaging adaptation type.
                TODO: Definition of this abstract messaging adaptation type and all its subtypes (and corresponding elements) is left for
                                        a future WS-Policy4MASC version. This has high priority.
                NOTE: Some of the possible subtypes are: merging, splitting, transforming, enriching and buffering of messages.
                !!!
    -->
        <xs:complexType name="MessagingAdaptation">
            <xs:complexContent>
                <xs:extension base="masc-pa:MASCConstruct">
                    <xs:sequence>
```

```
                                <xs:any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
                            </xs:sequence>
                        </xs:extension>
                        <!-- TODO: At this time, there is no real element or attribute here - the current specification is only
a placeholder (extensibility point). In the future, some real elements and/or attributes should be added. -->
                    </xs:complexContent>
                </xs:complexType>
                <!--
                        MessagingAdaptationRef (extends MASCRef): Definition of a reference to any messaging
adaptation.
    -->
                <xs:element name="MessagingAdaptationRef" type="masc-pa:MASCRef"/>
                <!--
                        !!!
                        PolicyAdaptation (extends MASCConstruct): Definition of an abstract policy adaptation type.
                                TODO: At this time, only activation and deactivation of policies are supported, along
with change of priority.
                                        Policy addition is performed by parsing a WS-Policy4MASC file.
                                        Policy deletion is not strictly necessary because policy deactivation
has similar effects.
                                        However, maybe some additional policy adaptations will be added
into a future version of WS-Policy4MASC.
                                NOTE: The adaptation (change) is performed ONLY for the policy alternative that is
currently used by the process instance
                                        for which this action is executed. However, this adaptation will be
also visible to all process instances using this
                                        policy alternative.
                        !!!
    -->
                <xs:complexType name="PolicyAdaptation">
                    <xs:complexContent>
                        <xs:extension base="masc-pa:MASCConstruct">
                            <xs:sequence>
                                <!-- The following element lists the affected policy assertions (goal, action, utility,
and/or meta-). -->
                                <xs:element ref="masc-pa:MASCPolicyAssertionRef" maxOccurs="unbounded"/>
                            </xs:sequence>
                        </xs:extension>
                    </xs:complexContent>
                </xs:complexType>
                <!--
                        PolicyAdaptationRef (extends MASCRef): Definition of a reference to any policy adaptation.
    -->
                <xs:element name="PolicyAdaptationRef" type="masc-pa:MASCRef"/>
                <!--
                        ActivatePolicies (extends PolicyAdaptation): Definition of an activation of a group of policies (that
were previously deactivated).
    -->
                <xs:element name="ActivatePolicyAssertions" type="masc-ap:ActivatePolicyAssertions">
                    <xs:annotation>
                        <xs:documentation>This element is intended for use as a WS-Policy policy
assertion</xs:documentation>
                    </xs:annotation>
                </xs:element>
                <xs:complexType name="ActivatePolicyAssertions">
                    <xs:complexContent>
                        <xs:extension base="masc-ap:PolicyAdaptation"/>
                        <!-- TODO: At this time, there is no element or attribute here. In the future, some elements and/or
attributes might be added. -->
                    </xs:complexContent>
                </xs:complexType>
                <!--
```

```
                ActivatePoliciesRef (extends MASCRef): Definition of a reference to an activation of a group of
policies

        (that were previously deactivated).
    -->
        <xs:element name="ActivatePolicyAssertionsRef" type="masc-pa:MASCRef"/>
        <!--
                DeactivatePolicies (extends PolicyAdaptation): Definition of a deactivation of a group of policies
(that were previously active).
    -->
        <xs:element name="DeactivatePolicyAssertions" type="masc-ap:DeactivatePolicyAssertions">
            <xs:annotation>
                <xs:documentation>This element is intended for use as a WS-Policy policy
assertion</xs:documentation>
            </xs:annotation>
        </xs:element>
        <xs:complexType name="DeactivatePolicyAssertions">
            <xs:complexContent>
                <xs:extension base="masc-ap:PolicyAdaptation"/>
                <!-- TODO: At this time, there is no element or attribute here. In the future, some elements and/or
attributes might be added. -->
            </xs:complexContent>
        </xs:complexType>
        <!--
                DeactivatePoliciesRef (extends MASCRef): Definition of a reference to a deactivation of a group
of policies

        (that were previously deactive).
    -->
        <xs:element name="DeactivatePolicyAssertionsRef" type="masc-pa:MASCRef"/>
        <!--
                ChangePolicyPriority (extends PolicyAdaptation): Definition of a change of policy priority for a
group of policies

                        (that were previously active).
    -->
        <xs:element name="ChangePolicyAssertionsPriority" type="masc-ap:ChangePolicyAssertionsPriority">
            <xs:annotation>
                <xs:documentation>This element is intended for use as a WS-Policy policy
assertion</xs:documentation>
            </xs:annotation>
        </xs:element>
        <xs:complexType name="ChangePolicyAssertionsPriority">
            <xs:complexContent>
                <xs:extension base="masc-ap:PolicyAdaptation">
                    <xs:attribute name="NewPriority" type="xs:positiveInteger" use="optional" default="1"/>
                </xs:extension>
                <!-- The following attribute specifies new priority - the higher the number, the higher the priority
(importance). -->
            </xs:complexContent>
        </xs:complexType>
        <!--
                ChangePolicyPriorityRef (extends MASCRef): Definition of a reference to a change of policy
priority for a group of policies

        (that were previously deactive).
    -->
        <xs:element name="ChangePolicyAssertionsPriorityRef" type="masc-pa:MASCRef"/>
        <!--
                !!!
                ActionCancellation (extends MASCConstruct): Definition of a cancellation of scheduled (future)
actions.
                                                                NOTE: For
```

```
example, if several retries of a current activity were scheduled, but only a few of them

        were performed (while the others are yet to be performed), then this construct can be

        used to cancel the remaining retries.
                        !!!
    -->
        <xs:element name="ActionCancellation" type="masc-ap:ActionCancellation">
                <xs:annotation>
                        <xs:documentation>This element is intended for use as a WS-Policy policy
assertion</xs:documentation>
                </xs:annotation>
        </xs:element>
        <xs:complexType name="ActionCancellation">
                <xs:complexContent>
                        <xs:extension base="masc-pa:MASCConstruct">
                                <xs:sequence>
                                        <!-- The following element is used to reference the action groups to be cancelled. -->
                                        <xs:element ref="masc-ap:ActionGroupRef" maxOccurs="unbounded"/>
                                </xs:sequence>
                        </xs:extension>
                </xs:complexContent>
        </xs:complexType>
        <!--
                        ActionCancellationRef (extends MASCRef): Definition of a reference to a cancellation of
scheduled (future) actions.
    -->
        <xs:element name="ActionCancellationRef" type="masc-pa:MASCRef"/>
        <!--
                        !!!
                        EventCancellation (extends MASCConstruct): Definition of making that an event is no longer
relevant.
                                                                                NOTE: For
example, after some handling (or some time) an event becomes no longer relevant,

        so it should no longer trigger any policies.
                        !!!
    -->
        <xs:element name="EventCancellation" type="masc-ap:EventCancellation">
                <xs:annotation>
                        <xs:documentation>This element is intended for use as a WS-Policy policy
assertion</xs:documentation>
                </xs:annotation>
        </xs:element>
        <xs:complexType name="EventCancellation">
                <xs:complexContent>
                        <xs:extension base="masc-pa:MASCConstruct">
                                <xs:sequence>
                                        <!-- The following element is used to reference the event(s) to be cancelled. -->
                                        <xs:element ref="masc-se:EventRef" maxOccurs="unbounded"/>
                                </xs:sequence>
                        </xs:extension>
                </xs:complexContent>
        </xs:complexType>
        <!--
                        EventCancellationRef (extends MASCRef): Definition of a reference to a making that an event is
no longer relevant.
    -->
        <xs:element name="EventCancellationRef" type="masc-pa:MASCRef"/>
</xs:schema>
```

Appendix 7: The File ws-policy4masc-up.xsd

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!-- Utility Policy Assertions Schema of WS-Policy4MASC version 0.8 -->
<!--Copyright (c) 2008 The University of New South Wales -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:tns="http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-up" xmlns:masc-up="http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-up" xmlns:masc-pa="http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-pa" xmlns:masc-se="http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-se" xmlns:masc-ex="http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-ex" targetNamespace="http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-up" elementFormDefault="qualified" attributeFormDefault="unqualified">
	<xs:import namespace="http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-pa" schemaLocation="ws-policy4masc-pa.xsd"/>
	<xs:import namespace="http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-ex" schemaLocation="ws-policy4masc-ex.xsd"/>
	<xs:import namespace="http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-se" schemaLocation="ws-policy4masc-se.xsd"/>
	<!--
		UtilityPolicyAssertion (extends MASCPolicyAssertion): Definition of a utility policy asssertion.

		TODO: To differentiate between different utility types, the following additional
			information might be useful:
				- schedules of future payments (currently there is only 1 payment at the time this utility policy assertion is calculated),
				- probability that an estimated future business value will be realized,
				- confidence in the precision of monetary estimates of intangible business values,
				- information about different types of intangible values that are represented.
			However, this is all left for future work.
	-->
	<xs:element name="UtilityPolicyAssertion" type="masc-up:UtilityPolicyAssertion">
		<xs:annotation>
			<xs:documentation>This element is intended for use as a WS-Policy policy assertion</xs:documentation>
		</xs:annotation>
	</xs:element>
	<xs:complexType name="UtilityPolicyAssertion">
		<xs:complexContent>
			<xs:extension base="masc-pa:MASCPolicyAssertion">
				<xs:sequence>
					<!-- The following element is used to reference the precondition for evaluation of the condition to be satisfied. -->
					<xs:element ref="masc-se:WhenRef"/>
					<!-- The following element is used to reference the amount to be paid. -->
					<xs:element ref="masc-ex:ArithmeticWithUnitExpressionRef"/>
				</xs:sequence>
				<xs:attribute name="BeneficiaryParty" type="xs:anyURI" use="required"/>
				<xs:attribute name="PayingParty" type="xs:anyURI" use="optional"/>
				<xs:attribute name="IsTangible" type="xs:boolean" use="optional" default="true"/>
			</xs:extension>
			<!-- The ManagementParty attribute (inherited from MASCPolicyAssertion) specifies the management party
				responsible for calculating and accounting this utility value. -->
			<!-- The following attribute specifies what party recives utility, i.e., benefits from it by receiving value (e.g., money).
				This is intended to be a name of a particular Web service or management party or
				one of the WS-Policy4MASC language constants (e.g., 'MASC_WSPROVIDER').
			-->
			<!-- The following attribute specifies what party is performing payment. If it is specified and not "MASC_NOAGREEDPAYER" ,
				then the specified utility policy refers to an agreed payment between two Web
```

```
services (and/or management parties).
                                    If it is not specified or it is specified but "MASC_NOAGREEDPAYER", then the
specified utility policy refers to an
                                    expected utility that the beneficiary party might receive from one or more other
parties.
                                    This is intended to be a name of a particular Web service or management party or
                                    one of the WS-Policy4MASC language constants (e.g., 'MASC_WSPROVIDER').
                                    Instead of "MASC_NOAGREEDPAYER", maybe an empty string ("") is a better
choice, if it can be specified for QName.
                                -->
                        <!-- default="MASC_NOAGREEDPAYER" -->
                        <!-- The following attribute can be used to specify whether the utility value is tangible (so this a
true monetary value)
                                    or intangible (so this is a monetary estimate of some non-monetary value). -->
                </xs:complexContent>
        </xs:complexType>
        <!--
                        UtilityPolicyAssertionRef (extends MASCRef): Definition of a reference to a utility policy
assertion.
    -->
        <xs:element name="UtilityPolicyAssertionRef" type="masc-pa:MASCRef"/>
</xs:schema>
```

Appendix 8: The File ws-policy4masc-mp.xsd

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!-- Metapolicy Assertions Schema of WS-Policy4MASC version 0.8 -->
<!--Copyright (c) 2006 The University of New South Wales -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:tns="http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-mp" xmlns:masc-mp="http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-mp" xmlns:masc-pa="http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-pa" xmlns:masc-ap="http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-ap" xmlns:masc-se="http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-se" xmlns:masc-ex="http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-ex" targetNamespace="http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-mp" elementFormDefault="qualified" attributeFormDefault="unqualified">
	<xs:import namespace="http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-pa" schemaLocation="ws-policy4masc-pa.xsd"/>
	<xs:import namespace="http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-ap" schemaLocation="ws-policy4masc-ap.xsd"/>
	<xs:import namespace="http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-se" schemaLocation="ws-policy4masc-se.xsd"/>
	<xs:import namespace="http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-ex" schemaLocation="ws-policy4masc-ex.xsd"/>
	<!--
		MetaPolicyAssertion (extends MASCPolicyAssertion): Definition of a metapolicy asssertion that is used to choose exactly 1
		action policy assertion between several (at least 2) that are conflicting
		because they are triggered at the same time.
		TODO: A number of additional strategies can be defined using the information that will be added into future utility policies.
		TODO: Specification of how more than 1 among the conflicting policy assertions are specified is left for future work.
	-->
	<xs:element name="MetaPolicyAssertion" type="masc-mp:MetaPolicyAssertion">
		<xs:annotation>
			<xs:documentation>This element is intended for use as a WS-Policy policy assertion</xs:documentation>
		</xs:annotation>
	</xs:element>
	<xs:complexType name="MetaPolicyAssertion">
		<xs:complexContent>
			<xs:extension base="masc-pa:MASCPolicyAssertion">
				<xs:sequence>
					<!-- The following element is used to reference at least 2 conflicting action policy assertions. -->
					<xs:element ref="masc-ap:ActionPolicyAssertionRef" minOccurs="2" maxOccurs="unbounded"/>
					<!-- The following optional element is used to reference the policy conflict resolution strategy.
					If no policy conflict resolution strategy is specified or it produces several equally valid options, then only the highest
					priority action policy assertion is executed (if priorities are specified at all). If even after this there are several equally
					valid options, then one of them is chosen at random (i.e., it is not determined which one will execute).
					TODO: Consider using the order of policy assertions in the policy repository to determine which one is executed if after
					conflict resolution strategy and/or priorities there are multiple valid options that can be chosen.
					-->
```

```
                        <xs:element ref="masc-mp:PolicyConflictResolutionStrategyRef" minOccurs="0"/>
                    </xs:sequence>
                    <xs:attribute name="PartyForWhichUtilityIsExamined" type="xs:anyURI"
use="required"/>
                </xs:extension>
            </xs:complexContent>
            <!-- The ManagementParty attribute (inherited from MASCPolicyAssertion) specifies the management
party
                        responsible for determining a policy conflict and applying the conflict resolution
strategy. -->
        </xs:complexType>
        <!--
                MetaPolicyAssertionRef (extends MASCRef): Definition of a reference to a metapolicy assertion.
    -->
        <xs:element name="MetaPolicyAssertionRef" type="masc-pa:MASCRef"/>
        <!--
                PolicyConflictResolutionStrategy: Definition of a policy conflict resolution strategy.

                NOTE: Addition of new policy conflict resolution strategies can be achieved by extending this
element type and/or
                        types of the contained elements.
                TODO: Make this a better extensiblity point (this would bring more flexibility at the cost of added
complexity).
                TODO: Consider whether it would be beneficial to require that default options must be specified
explicitly.
    -->
        <xs:element name="PolicyConflictResolutionStrategy" type="masc-mp:PolicyConflictResolutionStrategy"/>
        <xs:complexType name="PolicyConflictResolutionStrategy">
            <xs:complexContent>
                <xs:extension base="masc-pa:MASCConstruct">
                    <xs:sequence>
                        <!-- The following optional element is used to specify the ending condition until
which the monetary values are added.
                        If it is not specified, only immediate (direct) monetary implications of a considered action
policy assertion are added. -->
                        <xs:element ref="masc-mp:EndingCondition" minOccurs="0"/>
                        <!-- The following optional element is used to specify that estimated future business
values are NOT added together with
                        agreed payments, but are used as tiebreaker only. (For tiebreaking use the sum of
everything, NOT only the sum of estimated
                        future business values.)
                        If it is not specified, all agreed payments and estimated future business values are added.
                        (Note that it is illogical to add only estimated future value and using agreed payments as
tiebreakers. Therefore, this option
                        is not supported.) -->
                        <xs:element ref="masc-mp:EstimatedFutureValuesAsTiebreakerOnly"
minOccurs="0"/>
                        <!-- The following optional element is used to specify that either intangible or
tangible business values are NOT added (into a
                        calculation used for comparison between options), but are used as tiebreakers only. (For
tiebreaking use the sum of
                        everything, NOT only the sum of intangible or tangible values.)
                        If it is not specified, all tangible or intangible business values are added. -->
                        <xs:element ref="masc-mp:IntangibleOrTangibleValuesAsTiebreakerOnly"
minOccurs="0"/>
                        <!-- The following optional element is used to specify that the chosen option must
have limited costs.
                        If it is not specified, all options are considered. -->
                        <xs:element ref="masc-mp:LimitedCosts" minOccurs="0"/>
                    </xs:sequence>
                    <xs:attribute name="UseIntangibleValuesBeforeEstimatedFutureValues" type="xs:boolean"
default="true"/>
                </xs:extension>
```

```
            </xs:complexContent>
            <!-- The following attribute specifies for which party the utility values are taked into consideration. Utilty policy assertions where this
                party is neither the beneficiary party nor the paying party are not considered. (If this party is a paying party in some utility policy
                assertion, then this utility value gets inverted sign.)
                This is intended to be a name of a particular Web service or management party or one of the WS-Policy4MASC language
                constants (e.g., 'MASC_WSPROVIDER').
            -->
            <!-- The following attribute is used to specify whether intangible (or tangible) values or estimated future values are used as the first
                tiebreaker, if both dimensions are used a tiebreaker. If value of this attribute is "true", intangible (or tangible) values are the first
                tiebreaker. If it is "false", then estimated future values are the first tiebreaker. The former option is the default. -->
        </xs:complexType>
        <!--
                PolicyConflictResolutionStrategyRef (extends MASCRef): Definition of a reference to a policy conflict resolution strategy.
    -->
        <xs:element name="PolicyConflictResolutionStrategyRef" type="masc-pa:MASCRef"/>
        <!--
                EndingCondition: Definition of an ending condition - ending event and the maximal number of added events.

                TODO: Maybe it would be useful to make a reference to a When element instead of a simple event, but this is more
                    complicated for discrete event simulation. Therefore, this might be considered for future work.
    -->
        <xs:element name="EndingCondition" type="masc-mp:EndingCondition"/>
        <xs:complexType name="EndingCondition">
            <xs:sequence>
                <xs:element ref="masc-se:EventRef"/>
            </xs:sequence>
            <xs:attribute name="MaximalNumberOfAddedUtilityValues" type="xs:positiveInteger" use="required"/>
            <!-- The following attribute is needed to replace ending events that (for some reason) never occur. -->
        </xs:complexType>
        <!--
                EstimatedFutureValuesAsTiebreakerOnly: Definition of how estimated future business values are used as a tiebreaker.
    -->
        <xs:element name="EstimatedFutureValuesAsTiebreakerOnly" type="masc-mp:EstimatedFutureValuesAsTiebreakerOnly"/>
        <xs:complexType name="EstimatedFutureValuesAsTiebreakerOnly">
            <xs:sequence>
                <!-- The following element is used to reference the maximal difference between agreed payments that triggers use of
                    estimated future business values as tiebreaker. -->
                <xs:element ref="masc-ex:ArithmeticWithUnitExpressionRef"/>
            </xs:sequence>
        </xs:complexType>
        <!--
                IntangibleOrTangibleValuesAsTiebreakerOnly: Definition of how tangible or intangible business values are used as a tiebreaker.
    -->
        <xs:element name="IntangibleOrTangibleValuesAsTiebreakerOnly" type="masc-mp:IntangibleOrTangibleValuesAsTiebreakerOnly"/>
        <xs:complexType name="IntangibleOrTangibleValuesAsTiebreakerOnly">
            <xs:sequence>
                <!-- The following element is used to reference the maximal difference between tangible (intangible) business values that triggers
```

```
                        use of intangible (tangible) business values as tiebreaker. -->
                    <xs:element ref="masc-ex:ArithmeticWithUnitExpressionRef"/>
                </xs:sequence>
                <xs:attribute name="TangibleAreAddedWhileIntangibleAreTiebreaker" type="xs:boolean"
default="true"/>
                <!-- The following attribute is used to specify one of two possible combinations specified with the
containing element. If value of this
                     attribute is "true", then tangible values are added, while intangible values are used as a tiebreaker
only. If it is "false", then
                     intangible values are added, while tangible values are used as a tiebreaker only. The former option
is the default. -->
        </xs:complexType>
        <!--
                    LimitedCosts: Definition that costs of the chosen option must be limited.
    -->
        <xs:element name="LimitedCosts" type="masc-mp:LimitedCosts"/>
        <xs:complexType name="LimitedCosts">
                <xs:sequence>
                    <!-- The following element is used to reference the value of the cost limit.This value must be
negative. It is compared with the sum
                        of costs, which is always a negative number. If the sum of costs is less than the specified
limit value (NOT by an absolute
                        value, but by a value with the negative sign), an option is discarded.
                    -->
                    <xs:element ref="masc-ex:ArithmeticWithUnitExpressionRef"/>
                </xs:sequence>
        </xs:complexType>
</xs:schema>
```

Appendix 9: The File ws-policy4masc-se.xsd

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!-- States and Events Schema of WS-Policy4MASC version 0.8 -->
<!--Copyright (c) 2006 The University of New South Wales -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:tns="http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-se" xmlns:masc-se="http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-se" xmlns:masc-pa="http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-pa" xmlns:masc-ex="http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-ex" xmlns:masc-sc="http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-sc" targetNamespace="http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-se" elementFormDefault="qualified" attributeFormDefault="unqualified">
	<xs:import namespace="http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-pa" schemaLocation="ws-policy4masc-pa.xsd"/>
	<xs:import namespace="http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-ex" schemaLocation="ws-policy4masc-ex.xsd"/>
	<xs:import namespace="http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-sc" schemaLocation="ws-policy4masc-sc.xsd"/>
	<!-- TODO: Explore whether the type of the 'ID' attributes (in all definitions, but not references)
		should be 'NCName' instead of 'QName'!!! -->
	<!-- To specify when something should be performed, the following information is necessary -->
	<xs:element name="When" type="masc-se:When">
		<xs:annotation>
			<xs:documentation>This element is intended for use as a WS-Policy policy assertion
	</xs:documentation>
		</xs:annotation>
	</xs:element>
	<xs:complexType name="When">
		<xs:complexContent>
			<xs:extension base="masc-pa:MASCConstruct">
				<xs:sequence>
					<!-- Any of the listed states can lead to When being evaluated -->
					<xs:element ref="masc-se:AllowedStates"/>
					<!-- Any of the listed events can lead to When being evaluated.
		If a combination of events should be specified, then this should be specified in the BooleanExpression -->
					<xs:element ref="masc-se:PossibleTriggerEvents"/>
					<!-- The following optional Boolean expression specifies additional conditions to be satisfied.
		If it is not specified, the default is "true" (i.e., all conditions are satisfied).
					TODO: Modify the expressions schema to enable that occurence of an event and being in a particular state
			can be used within a Boolean expression. -->
					<xs:element ref="masc-ex:BooleanExpressionRef" minOccurs="0"/>
				</xs:sequence>
			</xs:extension>
		</xs:complexContent>
	</xs:complexType>
	<xs:element name="WhenRef" type="masc-pa:MASCRef"/>
	<xs:element name="AllowedStates">
		<xs:complexType>
			<xs:sequence>
				<xs:element ref="masc-se:StateRef" minOccurs="0" maxOccurs="unbounded"/>
			</xs:sequence>
		</xs:complexType>
	</xs:element>
	<xs:element name="PossibleTriggerEvents">
		<xs:complexType>
			<xs:sequence>
				<xs:element ref="masc-se:EventRef" maxOccurs="unbounded"/>
			</xs:sequence>
		</xs:complexType>
	</xs:element>
```

```
        <!-- Definition of states and state transitions -->
        <xs:element name="State" type="masc-se:State">
                <xs:annotation>
                        <xs:documentation>This element is intended for use as a WS-Policy policy assertion
        </xs:documentation>
                </xs:annotation>
        </xs:element>
        <xs:complexType name="State">
                <xs:complexContent>
                        <xs:extension base="masc-pa:MASCConstruct"/>
                </xs:complexContent>
        </xs:complexType>
        <xs:element name="StateRef" type="masc-pa:MASCRef"/>
        <xs:element name="StateTransition" type="masc-se:StateTransition">
                <xs:annotation>
                        <xs:documentation>This element is intended for use as a WS-Policy policy assertion
        </xs:documentation>
                </xs:annotation>
        </xs:element>
        <xs:complexType name="StateTransition">
                <xs:complexContent>
                        <xs:extension base="masc-pa:MASCConstruct">
                                <xs:sequence>
                                        <xs:element ref="masc-se:WhenRef"/>
                                        <!-- The following element specifies the name of the new state (since the old state is
specified in the When element) -->
                                        <xs:element ref="masc-se:StateRef"/>
                                </xs:sequence>
                        </xs:extension>
                </xs:complexContent>
        </xs:complexType>
        <xs:element name="StateTransitionRef" type="masc-pa:MASCRef"/>
        <!-- Definition of schedules -->
        <xs:element name="ScheduleDefinition" type="masc-se:ScheduleDefinition">
                <xs:annotation>
                        <xs:documentation>This element is intended for use as a WS-Policy policy
assertion</xs:documentation>
                </xs:annotation>
        </xs:element>
        <xs:complexType name="ScheduleDefinition">
                <xs:complexContent>
                        <xs:extension base="masc-pa:MASCConstruct">
                                <xs:sequence>
                                        <xs:choice>
                                                <xs:element ref="masc-se:StartTime"/>
                                                <xs:element ref="masc-se:StartImmediately"/>
                                        </xs:choice>
                                        <xs:element ref="masc-se:IntervalDuration"/>
                                        <!-- There are several ways to specify how the schedule ends.
                If more than one is specified, then the first one that occurs is used. -->
                                        <!-- If the following element is omitted, it is assumed that the schedule is not
delimited by particular date/time -->
                                        <xs:element ref="masc-se:StopTime" minOccurs="0"/>
                                        <!-- If the following element is omitted, it is assumed that the schedule is not
delimited by schedule duration -->
                                        <xs:element ref="masc-se:StopAfter" minOccurs="0"/>
                                        <!-- If the following element is omitted, it is assumed that the schedule is not
delimited by number of repetitions -->
                                        <xs:element ref="masc-se:MaxRepetitions" minOccurs="0"/>
                                        <!-- If the following element is omitted, it is assumed that the schedule is not
delimited by an even occuring -->
                                        <xs:element ref="masc-se:EventRef" minOccurs="0"/>
                                </xs:sequence>
```

```
                </xs:extension>
            </xs:complexContent>
        </xs:complexType>
        <xs:element name="ScheduleRef" type="masc-pa:MASCRef"/>
        <xs:element name="StartTime">
            <xs:complexType>
                <xs:attribute name="time" type="xs:dateTime" use="required"/>
            </xs:complexType>
        </xs:element>
        <xs:element name="StartImmediately"/>
        <xs:element name="IntervalDuration">
            <xs:complexType>
                <xs:attribute name="interval" type="xs:duration" use="required"/>
            </xs:complexType>
        </xs:element>
        <xs:element name="StopTime">
            <xs:complexType>
                <xs:attribute name="time" type="xs:dateTime" use="required"/>
            </xs:complexType>
        </xs:element>
        <xs:element name="StopAfter">
            <xs:complexType>
                <xs:attribute name="interval" type="xs:duration" use="required"/>
            </xs:complexType>
        </xs:element>
        <xs:element name="MaxRepetitions">
            <xs:complexType>
                <xs:attribute name="number" type="xs:positiveInteger" use="required"/>
            </xs:complexType>
        </xs:element>
        <!-- Definition of events - general information -->
        <xs:element name="EventDefinition" type="masc-se:EventDefinition">
            <xs:annotation>
                <xs:documentation>This element is intended for use as a WS-Policy policy
assertion</xs:documentation>
            </xs:annotation>
        </xs:element>
        <xs:complexType name="EventDefinition">
            <xs:complexContent>
                <xs:extension base="masc-pa:MASCConstruct">
                    <xs:choice>
                        <xs:element ref="masc-se:MonitoredDataItemUpdated"/>
                        <xs:element ref="masc-se:OperationInvocation"/>
                        <xs:element ref="masc-se:MessageReceipt"/>
                        <xs:element ref="masc-se:MessageToBeSent"/>
                        <xs:element ref="masc-se:MessageSending"/>
                        <xs:element ref="masc-se:GoalSatisfaction"/>
                        <xs:element ref="masc-se:ActionExecution"/>
                        <!-- The following element can be used for periodic events, period expiration, or
date/time based events -->
                        <xs:element ref="masc-se:ScheduledEventOccurence"/>
                        <xs:element ref="masc-se:ProcessInstanceStart"/>
                        <xs:element ref="masc-se:ProcessInstanceFinish"/>
                        <xs:element ref="masc-se:PartyToBeInitialized"/>
                        <xs:element ref="masc-se:PartyToBeFinalized"/>
                        <xs:element ref="masc-se:StateEntrance"/>
                        <xs:element ref="masc-se:StateExit"/>
                        <!-- The following element can be used for finer-grained control -->
                        <xs:element ref="masc-se:StateTransition"/>
                        <xs:element ref="masc-se:PolicyAssertionAddition"/>
                        <xs:element ref="masc-se:PolicyAssertionDeletion"/>
                    </xs:choice>
                </xs:extension>
```

```
            </xs:complexContent>
        </xs:complexType>
        <xs:element name="EventRef" type="masc-pa:MASCRef"/>
        <!-- Definition of various event types -->
        <!-- TODO: Make the language flexible in the way that additional new event types can be added. This can be achieved in a way
            analogous to WSOL 3.0 constraints and statements. -->
        <xs:element name="MonitoredDataItemUpdated">
            <xs:complexType>
                <xs:attribute name="ItemID" type="xs:QName" use="required"/>
            </xs:complexType>
        </xs:element>
        <xs:element name="OperationInvocation">
            <xs:complexType>
                <xs:sequence>
                    <xs:element ref="masc-sc:Scope-Operation"/>
                </xs:sequence>
                <xs:attribute name="Completed" type="xs:boolean" use="optional" default="true"/>
                <xs:attribute name="FaultsOccured" type="xs:boolean" use="optional" default="false"/>
            </xs:complexType>
        </xs:element>
        <xs:element name="MessageReceipt">
            <xs:complexType>
                <xs:sequence>
                    <xs:element ref="masc-sc:Scope-Message"/>
                </xs:sequence>
                <xs:attribute name="Completed" type="xs:boolean" use="optional" default="true"/>
            </xs:complexType>
        </xs:element>
        <xs:element name="MessageToBeSent">
            <xs:complexType>
                <xs:sequence>
                    <xs:element ref="masc-sc:Scope-Message"/>
                </xs:sequence>
                <xs:attribute name="Completed" type="xs:boolean" use="optional" default="true"/>
            </xs:complexType>
        </xs:element>
        <xs:element name="MessageSending">
            <xs:complexType>
                <xs:sequence>
                    <xs:element ref="masc-sc:Scope-Message"/>
                </xs:sequence>
                <xs:attribute name="Completed" type="xs:boolean" use="optional" default="true"/>
            </xs:complexType>
        </xs:element>
        <xs:element name="GoalSatisfaction">
            <xs:complexType>
                <xs:attribute name="GoalPolicyAssertionID" type="xs:QName" use="required"/>
                <xs:attribute name="Satisfied" type="xs:boolean" use="optional" default="true"/>
            </xs:complexType>
        </xs:element>
        <xs:element name="ActionExecution">
            <xs:complexType>
                <xs:attribute name="ActionPolicyAssertionID" type="xs:QName" use="required"/>
                <xs:attribute name="Completed" type="xs:boolean" use="optional" default="true"/>
            </xs:complexType>
        </xs:element>
        <xs:element name="ScheduledEventOccurence">
            <xs:complexType>
                <xs:attribute name="ScheduleID" type="xs:QName" use="required"/>
            </xs:complexType>
        </xs:element>
        <xs:element name="ProcessInstanceStart">
```

```
            <xs:complexType>
                <xs:attribute name="ProcessType" type="xs:QName" use="required"/>
                <xs:attribute name="Completed" type="xs:boolean" use="optional" default="true"/>
            </xs:complexType>
        </xs:element>
        <xs:element name="ProcessInstanceFinish">
            <xs:complexType>
                <xs:attribute name="ProcessType" type="xs:QName" use="required"/>
                <xs:attribute name="Completed" type="xs:boolean" use="optional" default="true"/>
            </xs:complexType>
        </xs:element>
        <xs:element name="PartyToBeInitialized">
            <xs:complexType>
                <xs:attribute name="PartyName" type="xs:QName" use="required"/>
                <!-- This is intended to be a name of a particular Web service or management party or
            one of the WS-Policy4MASC language constants (e.g., 'MASC-WSPROVIDER') -->
            </xs:complexType>
        </xs:element>
        <xs:element name="PartyToBeFinalized">
            <xs:complexType>
                <xs:attribute name="PartyName" type="xs:QName" use="required"/>
                <!-- This is intended to be a name of a particular Web service or management party or
            one of the WS-Policy4MASC language constants (e.g., 'MASC-WSPROVIDER') -->
            </xs:complexType>
        </xs:element>
        <xs:element name="StateEntrance">
            <xs:complexType>
                <xs:sequence>
                    <xs:element ref="masc-se:StateRef"/>
                </xs:sequence>
            </xs:complexType>
        </xs:element>
        <xs:element name="StateExit">
            <xs:complexType>
                <xs:sequence>
                    <xs:element ref="masc-se:StateRef"/>
                </xs:sequence>
            </xs:complexType>
        </xs:element>
        <xs:element name="StateTransitioning">
            <xs:complexType>
                <xs:sequence>
                    <xs:element ref="masc-se:StateTransitionRef"/>
                </xs:sequence>
            </xs:complexType>
        </xs:element>
        <xs:element name="PolicyAssertionAddition">
            <xs:complexType>
                <xs:attribute name="PolicyAssertionID" type="xs:QName" use="optional"/>
                <!-- If the following attribute is not specified, it is assumed that any policy assertion was added. -->
            </xs:complexType>
        </xs:element>
        <xs:element name="PolicyAssertionDeletion">
            <xs:complexType>
                <xs:attribute name="PolicyAssertionID" type="xs:QName" use="optional"/>
                <!-- If the following attribute is not specified, it is assumed that any policy assertion was deleted. --
>
            </xs:complexType>
        </xs:element>
</xs:schema>
```

Appendix 10: The File ws-policy4masc-sc.xsd

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!-- Scopes Schema of WS-Policy4MASC version 0.8 -->
<!--Copyright (c) 2008 The University of New South Wales -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:tns="http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-sc" xmlns:masc-sc="http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-sc" targetNamespace="http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-sc" elementFormDefault="qualified" attributeFormDefault="unqualified">
	<!-- The following are different WSDL scopes: service, operation, endpoint, message, and message part only (the other can be added later, if there is a need for them) -->
	<xs:element name="Scope-Service">
		<xs:complexType>
			<xs:sequence>
				<xs:element ref="masc-sc:ServiceName"/>
			</xs:sequence>
		</xs:complexType>
	</xs:element>
	<xs:element name="Scope-Endpoint">
		<xs:complexType>
			<xs:sequence>
				<xs:element ref="masc-sc:ServiceName"/>
				<xs:element ref="masc-sc:EndpointName"/>
			</xs:sequence>
		</xs:complexType>
	</xs:element>
	<xs:element name="Scope-Operation">
		<xs:complexType>
			<xs:sequence>
				<xs:element ref="masc-sc:ServiceName"/>
				<xs:element ref="masc-sc:EndpointName"/>
				<xs:element ref="masc-sc:OperationName"/>
			</xs:sequence>
		</xs:complexType>
	</xs:element>
	<xs:element name="Scope-Message">
		<xs:complexType>
			<xs:sequence>
				<xs:element ref="masc-sc:ServiceName"/>
				<xs:element ref="masc-sc:EndpointName"/>
				<xs:element ref="masc-sc:OperationName"/>
				<xs:choice>
					<xs:element ref="masc-sc:MessageName"/>
					<xs:element name="InputMessage"/>
					<xs:element name="OutputMessage"/>
				</xs:choice>
			</xs:sequence>
		</xs:complexType>
	</xs:element>
	<xs:element name="Scope-MessagePart">
		<xs:complexType>
			<xs:sequence>
				<xs:element ref="masc-sc:ServiceName"/>
				<xs:element ref="masc-sc:EndpointName"/>
				<xs:element ref="masc-sc:OperationName"/>
				<xs:choice>
					<xs:element ref="masc-sc:MessageName"/>
					<xs:element name="InputMessage"/>
					<xs:element name="OutputMessage"/>
					<xs:element name="AnyFaultMessage"/>
				</xs:choice>
				<xs:element ref="masc-sc:MessagePartName"/>
```

```
                </xs:sequence>
            </xs:complexType>
        </xs:element>
        <!-- The following are references to various WSDL constructs -->
        <xs:element name="ServiceName">
            <xs:complexType>
                <xs:attribute name="Name" type="xs:QName" use="required"/>
            </xs:complexType>
        </xs:element>
        <xs:element name="EndpointName">
            <xs:complexType>
                <xs:attribute name="Name" type="xs:QName" use="required"/>
            </xs:complexType>
        </xs:element>
        <xs:element name="OperationName">
            <xs:complexType>
                <xs:attribute name="Name" type="xs:QName" use="required"/>
            </xs:complexType>
        </xs:element>
        <xs:element name="MessageName">
            <xs:complexType>
                <xs:attribute name="Name" type="xs:QName" use="required"/>
            </xs:complexType>
        </xs:element>
        <xs:element name="MessagePartName">
            <xs:complexType>
                <xs:attribute name="Name" type="xs:QName" use="required"/>
            </xs:complexType>
        </xs:element>
        <!-- The following are the language constants that can be used instead of particular names in the above
references
        TODO: Semantic rules related to combinations of these constants with general references have to be built into
language tools
        TODO: If appropriate, add constants for 'All' and 'Many' (in addition to current 'Any') -->
        <xs:element name="MASC-ANYSERVICE"/>
        <xs:element name="MASC-ANYENDPOINT"/>
        <xs:element name="MASC-ANYOPERATION"/>
        <xs:element name="MASC-ANYMESSAGE"/>
        <!-- It does not seem meaningful to specify "MASC-ANYMESSAGEPART" due to the diversity of message
parts -->
</xs:schema>
```

Appendix 11: The File ws-policy4masc-om.xsd

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!-- Ontological Meaning Schema of WS-Policy4MASC version 0.8 -->
<!--Copyright (c) 2006 The University of New South Wales -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:tns="http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-om" xmlns:masc-om="http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-om" xmlns:masc-pa="http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-pa" targetNamespace="http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-om" elementFormDefault="qualified" attributeFormDefault="unqualified">
	<xs:import namespace="http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-pa" schemaLocation="ws-policy4masc-pa.xsd"/>
	<!--
			OntologicalMeaning (extends MASCConstruct): Definition of an ontological meaning.
			NOTE: At this time, ontological meaning is used only for comparisons. In principle, it can be used as configuration data
		for monitoring/calculation, but this might be too complicated. Therefore, additional element <masc-ap:ConfigurationData> is
		specified for the element <masc-ap:MonitoredDataItemCollection> in this version of the WS-Policy4MASC language.
	-->
	<xs:element name="OntologicalMeaning" type="masc-om:OntologicalMeaning">
		<xs:annotation>
			<xs:documentation>This element is intended for use as a WS-Policy policy assertion</xs:documentation>
		</xs:annotation>
	</xs:element>
	<xs:complexType name="OntologicalMeaning">
		<xs:complexContent>
			<xs:extension base="masc-pa:MASCConstruct">
				<xs:attribute name="OntologicalDefinition" type="xs:QName" use="optional"/>
				<xs:attribute name="OntologicalLanguage" type="xs:anyURI" use="optional" default="http://masc.web.cse.unsw.edu.au/Schemas/OntologySchema.xsd"/>
			</xs:extension>
			<!-- This attribute refers to a namespace and string name within this namespace that can be used to find an ontological definition of the used concept. -->
			<!-- This attribute refers to an URI of the definition of the language in which the ontolological definition is defined. -->
		</xs:complexContent>
	</xs:complexType>
	<!--
			OntologicalMeaningRef (extends MASCRef): Definition of a reference to an ontological meaning.
	-->
	<xs:element name="OntologicalMeaningRef" type="masc-pa:MASCRef"/>
</xs:schema>
```

Appendix 12: The File ws-policy4masc-ex.xsd

I) The "new, more flexible, but far from complete" version:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!-- Expressions Schema of WS-Policy4MASC version 0.8 -->
<!--Copyright (c) 2006 The University of New South Wales -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:tns="http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-ex" xmlns:masc-ex="http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-ex" xmlns:masc-pa="http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-pa"
targetNamespace="http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-ex" elementFormDefault="qualified"
attributeFormDefault="unqualified">
	<xs:import namespace="http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-pa" schemaLocation="ws-policy4masc-pa.xsd"/>
	<!--
		BooleanConstant: Definition of a Boolean constant.
	-->
	<xs:element name="BooleanConstant" type="xs:boolean"/>
	<!--
		BooleanExpression: Definition of a Boolean expression
	-->
	<xs:element name="BooleanExpression">
		<xs:complexType>
			<xs:choice>
				<xs:element ref="masc-ex:BooleanConstant"/>
			</xs:choice>
			<xs:attribute name="MASCID" type="xs:NCName" use="required"/>
		</xs:complexType>
	</xs:element>
	<!--
		BooleanExpressionRef: Definition of a reference to a boolean expression.
	-->
	<xs:element name="BooleanExpressionRef" type="masc-pa:MASCRef"/>
	<!--
		ArithmeticWithUnitConstant: Definition of an arithmetic with unit constant.
	-->
	<xs:element name="ArithmeticWithUnitConstant" type="masc-ex:ArithmeticWithUnitConstant"/>
	<xs:complexType name="ArithmeticWithUnitConstant">
		<xs:sequence>
			<xs:element ref="masc-ex:ArithmeticValue"/>
			<xs:element ref="masc-ex:Unit"/>
		</xs:sequence>
	</xs:complexType>
	<xs:element name="ArithmeticValue" type="xs:double"/>
	<xs:element name="Unit" type="masc-ex:Unit"/>
	<xs:complexType name="Unit">
		<xs:attribute name="OntologicalType" type="xs:QName" use="required"/>
	</xs:complexType>
	<!--
		ArithmeticWithUnitExpression: Definition of an arithmetic with unit expression.
	-->
	<xs:element name="ArithmeticWithUnitExpression">
		<xs:complexType>
			<xs:choice>
				<xs:element ref="masc-ex:ArithmeticWithUnitConstant"/>
			</xs:choice>
			<xs:attribute name="MASCID" type="xs:NCName" use="required"/>
		</xs:complexType>
	</xs:element>
	<!--
		ArithmeticWithUnitExpressionRef: Definition of a reference to an arithmetic with unit expression.
	-->
```

```
	<xs:element name="ArithmeticWithUnitExpressionRef" type="masc-pa:MASCRef"/>
	<!--
			VariableRef: Definition of a reference type to a process variable (in the base process or the
variation process).
  -->
	<xs:complexType name="ProcessVariableRef">
		<xs:attribute name="ReferenceType" type="xs:string" default="XPath"/>
		<xs:attribute name="ValueDataType" type="xs:QName" use="required"/>
		<!-- The following attribute could be used to denote qualified name of the process type. However, it
introduces unwanted
			redundances (e.g., with policy attachment) and potential errors, so it is commented out (at least for
now). - ->
			<xs:attribute name="ProcessRef" type="xs:QName"/>
		-->
		<!-- The following element is used to specify what type of format is used to specify the
			content of an element of the encompassing complexType type. -->
		<!-- The argument for 'ValueDataType' can be the an xsi:type, but also a complex type. -->
	</xs:complexType>
</xs:schema>
```

II) The “old, complete, but cumbersome” version:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Expressions Schema of WS-Policy4MASC version 0.8 -->
<!--Copyright (c) 2008 The University of New South Wales -->
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:tns="http://www.sce.unsw.edu.au/~vtosic/ws-
policy4masc/masc-ex" xmlns:masc-ex="http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-ex"
targetNamespace="http://www.sce.unsw.edu.au/~vtosic/ws-policy4masc/masc-ex" elementFormDefault="qualified"
attributeFormDefault="unqualified">
	<!-- TODO: This big schema probably should be broken into a number of smaller schemas.
	TODO: This schema should be updated to use the same conventions as the other WS-Policy4MASC schemas.
	TODO: This schema must be extended to enable checks whether an event was raised and checks what is the
current state. -->
	<!-- booleanExpression -->
	<xsd:element name="booleanExpression" type="masc-ex:booleanExpressionType"/>
	<xsd:complexType name="booleanExpressionType">
		<xsd:choice>
			<xsd:element ref="masc-ex:booleanConstant"/>
			<xsd:element ref="masc-ex:booleanVariable"/>
			<xsd:element ref="masc-ex:externalOperationResult"/>
			<xsd:element ref="masc-ex:booleanExpression"/>
			<xsd:element ref="masc-ex:quantifiedExpression"/>
			<xsd:element ref="masc-ex:monitoringCheckExpression"/>
			<xsd:element ref="masc-ex:BooleanExpressionRef"/>
			<xsd:element ref="masc-ex:unsatisfiedConstraintRef"/>
			<xsd:sequence>
				<xsd:element ref="masc-ex:unaryBooleanOperator"/>
				<xsd:element ref="masc-ex:booleanExpression"/>
			</xsd:sequence>
			<xsd:sequence>
				<xsd:element ref="masc-ex:binaryBooleanOperator"/>
				<xsd:element ref="masc-ex:booleanExpression"/>
				<xsd:element ref="masc-ex:booleanExpression"/>
			</xsd:sequence>
			<xsd:sequence>
				<xsd:element ref="masc-ex:booleanComparator"/>
				<xsd:element ref="masc-ex:booleanExpression"/>
				<xsd:element ref="masc-ex:booleanExpression"/>
			</xsd:sequence>
			<xsd:sequence>
				<xsd:element ref="masc-ex:arithmeticComparator"/>
				<xsd:element ref="masc-ex:arithmeticExpression"/>
				<xsd:element ref="masc-ex:arithmeticExpression"/>
```

```
                </xsd:sequence>
                <xsd:sequence>
                        <xsd:element ref="masc-ex:arithmeticWithUnitComparator"/>
                        <xsd:element ref="masc-ex:arithmeticWithUnitExpression"/>
                        <xsd:element ref="masc-ex:arithmeticWithUnitExpression"/>
                </xsd:sequence>
                <xsd:sequence>
                        <xsd:element ref="masc-ex:stringComparator"/>
                        <xsd:element ref="masc-ex:stringExpression"/>
                        <xsd:element ref="masc-ex:stringExpression"/>
                </xsd:sequence>
                <xsd:sequence>
                        <xsd:element ref="masc-ex:timeComparator"/>
                        <xsd:element ref="masc-ex:timeExpression"/>
                        <xsd:element ref="masc-ex:timeExpression"/>
                </xsd:sequence>
                <xsd:sequence>
                        <xsd:element ref="masc-ex:isMonitoredQoSMetric"/>
                        <xsd:element ref="masc-ex:qosMetric"/>
                </xsd:sequence>
                <xsd:sequence>
                        <xsd:element ref="masc-ex:isMonitoredContextProperty"/>
                        <xsd:element ref="masc-ex:contextProperty"/>
                </xsd:sequence>
        </xsd:choice>
        <xsd:attribute name="name" type="xsd:NCName" use="optional"/>
</xsd:complexType>
<!-- booleanConstant -->
<xsd:element name="booleanConstant" type="masc-ex:booleanConstantType"/>
<xsd:complexType name="booleanConstantType">
        <xsd:attribute name="type" type="masc-ex:booleanConstantTypeChoice" use="required"/>
</xsd:complexType>
<!--booleanConstantTypeChoice -->
<xsd:simpleType name="booleanConstantTypeChoice">
        <xsd:restriction base="xsd:string">
                <xsd:enumeration value="true"/>
                <xsd:enumeration value="false"/>
        </xsd:restriction>
</xsd:simpleType>
<!-- unaryBooleanOperatorTypeChoice -->
<xsd:simpleType name="unaryBooleanOperatorTypeChoice">
        <xsd:restriction base="xsd:string">
                <xsd:enumeration value="NOT"/>
        </xsd:restriction>
</xsd:simpleType>
<!-- binaryBooleanOperatorTypeChoice -->
<xsd:simpleType name="binaryBooleanOperatorTypeChoice">
        <xsd:restriction base="xsd:string">
                <xsd:enumeration value="AND"/>
                <xsd:enumeration value="OR"/>
                <xsd:enumeration value="XOR"/>
                <xsd:enumeration value="EQUIVALENT"/>
                <xsd:enumeration value="IMPLIES"/>
        </xsd:restriction>
</xsd:simpleType>
<!-- booleanComparatorTypeChoice -->
<xsd:simpleType name="booleanComparatorTypeChoice">
        <xsd:restriction base="xsd:string">
                <xsd:enumeration value="=="/>
                <xsd:enumeration value="!="/>
        </xsd:restriction>
</xsd:simpleType>
<!-- quantifiedExpressionTypeChoice -->
```

```
<xsd:simpleType name="quantifiedExpressionTypeChoice">
	<xsd:restriction base="xsd:string">
		<xsd:enumeration value="FOR_ALL_IN"/>
		<xsd:enumeration value="EXISTS_IN"/>
	</xsd:restriction>
</xsd:simpleType>
<!-- unaryArithmeticOperatorTypeChoice -->
<xsd:simpleType name="unaryArithmeticOperatorTypeChoice">
	<xsd:restriction base="xsd:string">
		<xsd:enumeration value="-"/>
	</xsd:restriction>
</xsd:simpleType>
<!-- binaryArithmeticOperatorTypeChoice -->
<xsd:simpleType name="binaryArithmeticOperatorTypeChoice">
	<xsd:restriction base="xsd:string">
		<xsd:enumeration value="+"/>
		<xsd:enumeration value="-"/>
		<xsd:enumeration value="*"/>
		<xsd:enumeration value="/"/>
		<xsd:enumeration value="**"/>
	</xsd:restriction>
</xsd:simpleType>
<!-- arithmeticComparatorTypeChoice -->
<xsd:simpleType name="arithmeticComparatorTypeChoice">
	<xsd:restriction base="xsd:string">
		<xsd:enumeration value="&lt;"/>
		<xsd:enumeration value=">"/>
		<xsd:enumeration value="=="/>
		<xsd:enumeration value="&lt;="/>
		<xsd:enumeration value=">="/>
		<xsd:enumeration value="!="/>
	</xsd:restriction>
</xsd:simpleType>
<!-- arithmeticWithUnitComparatorTypeChoice -->
<xsd:simpleType name="arithmeticWithUnitComparatorTypeChoice">
	<xsd:restriction base="xsd:string">
		<xsd:enumeration value="&lt;"/>
		<xsd:enumeration value=">"/>
		<xsd:enumeration value="=="/>
		<xsd:enumeration value="&lt;="/>
		<xsd:enumeration value=">="/>
		<xsd:enumeration value="!="/>
	</xsd:restriction>
</xsd:simpleType>
<!-- stringComparatorTypeChoice -->
<xsd:simpleType name="stringComparatorTypeChoice">
	<xsd:restriction base="xsd:string">
		<xsd:enumeration value="&lt;"/>
		<xsd:enumeration value=">"/>
		<xsd:enumeration value="=="/>
		<xsd:enumeration value="&lt;="/>
		<xsd:enumeration value=">="/>
		<xsd:enumeration value="!="/>
	</xsd:restriction>
</xsd:simpleType>
<!-- timeOperatorTypeChoice -->
<xsd:simpleType name="timeOperatorTypeChoice">
	<xsd:restriction base="xsd:string">
		<xsd:enumeration value="BEFORE"/>
		<xsd:enumeration value="AFTER"/>
		<xsd:enumeration value="DURING"/>
		<xsd:enumeration value="BETWEEN"/>
	</xsd:restriction>
```

```
</xsd:simpleType>
<!-- timeComparatorTypeChoice -->
<xsd:simpleType name="timeComparatorTypeChoice">
	<xsd:restriction base="xsd:string">
		<xsd:enumeration value="&lt;"/>
		<xsd:enumeration value=">"/>
		<xsd:enumeration value="=="/>
		<xsd:enumeration value="&lt;="/>
		<xsd:enumeration value=">="/>
		<xsd:enumeration value="!="/>
	</xsd:restriction>
</xsd:simpleType>
<!-- binaryArithmeticWithUnitOperator1TypeChoice -->
<xsd:simpleType name="binaryArithmeticWithUnitOperator1TypeChoice">
	<xsd:restriction base="xsd:string">
		<xsd:enumeration value="+"/>
		<xsd:enumeration value="-"/>
	</xsd:restriction>
</xsd:simpleType>
<!-- binaryArithmeticWithUnitOperator2TypeChoice -->
<xsd:simpleType name="binaryArithmeticWithUnitOperator2TypeChoice">
	<xsd:restriction base="xsd:string">
		<xsd:enumeration value="*"/>
		<xsd:enumeration value="/"/>
	</xsd:restriction>
</xsd:simpleType>
<!-- externalOperationResult -->
<xsd:element name="externalOperationResult" type="masc-ex:externalOperationResultType"/>
<xsd:complexType name="externalOperationResultType">
	<xsd:attribute name="callID" type="xsd:QName" use="required"/>
	<xsd:attribute name="resultPartName" type="xsd:QName" use="required"/>
</xsd:complexType>
<!-- unaryBooleanOperator -->
<xsd:element name="unaryBooleanOperator" type="masc-ex:unaryBooleanOperatorType"/>
<xsd:complexType name="unaryBooleanOperatorType">
	<xsd:attribute name="type" type="masc-ex:unaryBooleanOperatorTypeChoice" use="required"/>
</xsd:complexType>
<!-- binaryBooleanOperator -->
<xsd:element name="binaryBooleanOperator" type="masc-ex:binaryBooleanOperatorType"/>
<xsd:complexType name="binaryBooleanOperatorType">
	<xsd:attribute name="type" type="masc-ex:binaryBooleanOperatorTypeChoice" use="required"/>
</xsd:complexType>
<!-- booleanComparator -->
<xsd:element name="booleanComparator" type="masc-ex:booleanComparatorType"/>
<xsd:complexType name="booleanComparatorType">
	<xsd:attribute name="type" type="masc-ex:booleanComparatorTypeChoice" use="required"/>
</xsd:complexType>
<!-- quantifiedExpression -->
<xsd:element name="quantifiedExpression" type="masc-ex:quantifiedExpressionType"/>
<xsd:complexType name="quantifiedExpressionType">
	<xsd:choice>
		<xsd:element ref="masc-ex:booleanExpression"/>
		<xsd:element ref="masc-ex:quantifiedExpressionRef"/>
	</xsd:choice>
	<xsd:attribute name="name" type="xsd:NCName" use="optional"/>
	<xsd:attribute name="type" type="masc-ex:quantifiedExpressionTypeChoice" use="required"/>
	<xsd:attribute name="arrayVariableName" type="xsd:QName" use="required"/>
</xsd:complexType>
<!-- isMonitoredQoSMetric -->
<xsd:element name="isMonitoredQoSMetric" type="masc-ex:isMonitoredQoSMetricType"/>
<xsd:complexType name="isMonitoredQoSMetricType"/>
<!-- isMonitoredContextProperty -->
<xsd:element name="isMonitoredContextProperty" type="masc-ex:isMonitoredContextPropertyType"/>
```

```
        <xsd:complexType name="isMonitoredContextPropertyType"/>
        <!-- monitoringCheckExpression -->
        <xsd:element name="monitoringCheckExpression" type="masc-ex:monitoringCheckExpressionType"/>
        <xsd:complexType name="monitoringCheckExpressionType">
                <xsd:choice>
                        <xsd:sequence>
                                <xsd:element ref="masc-ex:isMonitoredQoSMetric"/>
                                <xsd:element ref="masc-ex:qosMetric"/>
                        </xsd:sequence>
                        <xsd:sequence>
                                <xsd:element ref="masc-ex:isMonitoredContextProperty"/>
                                <xsd:element ref="masc-ex:contextProperty"/>
                        </xsd:sequence>
                        <xsd:element ref="masc-ex:monitoringCheckExpressionRef"/>
                </xsd:choice>
                <xsd:attribute name="name" type="xsd:NCName" use="optional"/>
        </xsd:complexType>
        <!-- arithmeticExpression -->
        <xsd:element name="arithmeticExpression" type="masc-ex:arithmeticExpressionType"/>
        <xsd:complexType name="arithmeticExpressionType">
                <xsd:choice>
                        <xsd:element ref="masc-ex:arithmeticConstant"/>
                        <xsd:element ref="masc-ex:arithmeticVariable"/>
                        <xsd:element ref="masc-ex:externalOperationResult"/>
                        <xsd:element ref="masc-ex:arithmeticExpression"/>
                        <xsd:element ref="masc-ex:arithmeticExpressionRef"/>
                        <xsd:sequence>
                                <xsd:element ref="masc-ex:unaryArithmeticOperator"/>
                                <xsd:element ref="masc-ex:arithmeticExpression"/>
                        </xsd:sequence>
                        <xsd:sequence>
                                <xsd:element ref="masc-ex:binaryArithmeticOperator"/>
                                <xsd:element ref="masc-ex:arithmeticExpression"/>
                                <xsd:element ref="masc-ex:arithmeticExpression"/>
                        </xsd:sequence>
                </xsd:choice>
                <xsd:attribute name="name" type="xsd:NCName" use="optional"/>
                <!--<xsd:element ref="arithmeticFunctionCall"/>-->
        </xsd:complexType>
        <!-- arithmeticConstant -->
        <xsd:element name="arithmeticConstant" type="masc-ex:arithmeticConstantType"/>
        <xsd:complexType name="arithmeticConstantType">
                <xsd:choice>
                        <xsd:element ref="masc-ex:integerConstant"/>
                        <xsd:element ref="masc-ex:floatConstant"/>
                        <xsd:element ref="masc-ex:doubleConstant"/>
                        <xsd:element ref="masc-ex:longConstant"/>
                </xsd:choice>
        </xsd:complexType>
        <!-- qoSMetric -->
        <xsd:element name="qosMetric" type="masc-ex:qosMetricType"/>
        <xsd:complexType name="qosMetricType">
                <xsd:sequence>
                        <xsd:element ref="masc-ex:usedQoSmetric" minOccurs="0" maxOccurs="unbounded"/>
                </xsd:sequence>
                <xsd:attribute name="metricType" type="xsd:QName" use="required"/>
                <xsd:attribute name="metricUnit" type="xsd:QName" use="required"/>
                <xsd:attribute name="service" type="xsd:QName" use="required"/>
                <xsd:attribute name="portOrPortType" type="xsd:QName" use="required"/>
                <xsd:attribute name="operation" type="xsd:QName" use="required"/>
                <xsd:attribute name="monitoredFor" type="xsd:anyURI" use="optional" default="MASC-ROLE-
PROVIDER"/>
                <xsd:attribute name="monitoredBy" type="xsd:anyURI" use="required"/>
```

```
            <xsd:attribute name="pushed" type="xsd:boolean" use="optional" default="true"/>
        </xsd:complexType>
        <!-- usedQoSmetric -->
        <xsd:element name="usedQoSmetric" type="masc-ex:usedQoSmetricType"/>
        <xsd:complexType name="usedQoSmetricType">
            <xsd:attribute name="metricType" type="xsd:QName" use="required"/>
            <xsd:attribute name="metricUnit" type="xsd:QName" use="required"/>
            <xsd:attribute name="service" type="xsd:QName" use="required"/>
            <xsd:attribute name="portOrPortType" type="xsd:QName" use="required"/>
            <xsd:attribute name="operation" type="xsd:QName" use="required"/>
            <xsd:attribute name="monitoredFor" type="xsd:anyURI" use="optional" default="MASC-ROLE-
PROVIDER"/>
            <xsd:attribute name="monitoredBy" type="xsd:anyURI" use="required"/>
            <xsd:attribute name="monitoringSchedule" type="xsd:QName" use="optional"/>
            <xsd:attribute name="exchangeSchedule" type="xsd:QName" use="optional"/>
            <xsd:attribute name="pushed" type="xsd:boolean" use="optional" default="true"/>
        </xsd:complexType>
        <!-- contextProperty -->
        <xsd:element name="contextProperty" type="masc-ex:contextPropertyType"/>
        <xsd:complexType name="contextPropertyType">
            <xsd:sequence>
                <xsd:element ref="masc-ex:usedContextProperty" minOccurs="0" maxOccurs="unbounded"/>
            </xsd:sequence>
            <xsd:attribute name="propertyType" type="xsd:QName" use="required"/>
            <xsd:attribute name="propertyUnit" type="xsd:QName" use="required"/>
            <xsd:attribute name="service" type="xsd:QName" use="required"/>
            <xsd:attribute name="monitoredFor" type="xsd:anyURI" use="optional" default="MASC-ROLE-
PROVIDER"/>
            <xsd:attribute name="monitoredBy" type="xsd:anyURI" use="required"/>
            <xsd:attribute name="monitoringSchedule" type="xsd:QName" use="optional"/>
            <xsd:attribute name="exchangeSchedule" type="xsd:QName" use="optional"/>
            <xsd:attribute name="pushed" type="xsd:boolean" use="optional" default="true"/>
        </xsd:complexType>
        <!-- usedContextProperty -->
        <xsd:element name="usedContextProperty" type="masc-ex:usedContextPropertyType"/>
        <xsd:complexType name="usedContextPropertyType">
            <xsd:attribute name="propertyType" type="xsd:QName" use="required"/>
            <xsd:attribute name="propertyUnit" type="xsd:QName" use="required"/>
            <xsd:attribute name="service" type="xsd:QName" use="required"/>
            <xsd:attribute name="monitoredFor" type="xsd:anyURI" use="optional" default="MASC-ROLE-
PROVIDER"/>
            <xsd:attribute name="monitoredBy" type="xsd:anyURI" use="required"/>
            <xsd:attribute name="monitoringSchedule" type="xsd:QName" use="optional"/>
            <xsd:attribute name="exchangeSchedule" type="xsd:QName" use="optional"/>
            <xsd:attribute name="pushed" type="xsd:boolean" use="optional" default="true"/>
        </xsd:complexType>
        <!-- arithmeticWithUnitExpression -->
        <xsd:element name="arithmeticWithUnitExpression" type="masc-ex:arithmeticWithUnitExpressionType"/>
        <xsd:complexType name="arithmeticWithUnitExpressionType">
            <xsd:choice>
                <xsd:element ref="masc-ex:arithmeticWithUnitVariable"/>
                <xsd:element ref="masc-ex:qosMetric"/>
                <xsd:element ref="masc-ex:contextProperty"/>
                <xsd:element ref="masc-ex:arithmeticWithUnitExpressionRef"/>
                <xsd:sequence>
                    <xsd:element ref="masc-ex:binaryArithmeticWithUnitOperator1"/>
                    <xsd:element ref="masc-ex:arithmeticWithUnitExpression"/>
                    <xsd:element ref="masc-ex:arithmeticWithUnitExpression"/>
                </xsd:sequence>
                <xsd:sequence>
                    <xsd:element ref="masc-ex:binaryArithmeticWithUnitOperator2"/>
                    <xsd:element ref="masc-ex:arithmeticWithUnitExpression"/>
                    <xsd:element ref="masc-ex:arithmeticConstant"/>
```

```
                </xsd:sequence>
                <xsd:element ref="masc-ex:numberWithUnitConstant"/>
            </xsd:choice>
            <xsd:attribute name="name" type="xsd:NCName" use="optional"/>
            <!--<xsd:element ref="arithmeticFunctionCall"/>-->
        </xsd:complexType>
        <!-- unaryArithmeticOperator -->
        <xsd:element name="unaryArithmeticOperator" type="masc-ex:unaryArithmeticOperatorType"/>
        <xsd:complexType name="unaryArithmeticOperatorType">
            <xsd:attribute name="type" type="masc-ex:unaryArithmeticOperatorTypeChoice" use="required"/>
        </xsd:complexType>
        <!-- binaryArithmeticOperator -->
        <xsd:element name="binaryArithmeticOperator" type="masc-ex:binaryArithmeticOperatorType"/>
        <xsd:complexType name="binaryArithmeticOperatorType">
            <xsd:attribute name="type" type="masc-ex:binaryArithmeticOperatorTypeChoice" use="required"/>
        </xsd:complexType>
        <!-- binaryArithmeticWithUnitOperator1 -->
        <xsd:element name="binaryArithmeticWithUnitOperator1" type="masc-
ex:binaryArithmeticWithUnitOperator1Type"/>
        <xsd:complexType name="binaryArithmeticWithUnitOperator1Type">
            <xsd:attribute name="type" type="masc-ex:binaryArithmeticWithUnitOperator1TypeChoice"
use="required"/>
        </xsd:complexType>
        <!-- binaryArithmeticWithUnitOperator2 -->
        <xsd:element name="binaryArithmeticWithUnitOperator2" type="masc-
ex:binaryArithmeticWithUnitOperator2Type"/>
        <xsd:complexType name="binaryArithmeticWithUnitOperator2Type">
            <xsd:attribute name="type" type="masc-ex:binaryArithmeticWithUnitOperator2TypeChoice"
use="required"/>
        </xsd:complexType>
        <!-- arithmeticComparator -->
        <xsd:element name="arithmeticComparator" type="masc-ex:arithmeticComparatorType"/>
        <xsd:complexType name="arithmeticComparatorType">
            <xsd:attribute name="type" type="masc-ex:arithmeticComparatorTypeChoice" use="required"/>
        </xsd:complexType>
        <!-- arithmeticWithUnitComparator -->
        <xsd:element name="arithmeticWithUnitComparator" type="masc-ex:arithmeticWithUnitComparatorType"/>
        <xsd:complexType name="arithmeticWithUnitComparatorType">
            <xsd:attribute name="type" type="masc-ex:arithmeticWithUnitComparatorTypeChoice" use="required"/>
        </xsd:complexType>
        <!-- stringExpression -->
        <xsd:element name="stringExpression" type="masc-ex:stringExpressionType"/>
        <xsd:complexType name="stringExpressionType">
            <xsd:choice>
                <xsd:element ref="masc-ex:stringConstant"/>
                <xsd:element ref="masc-ex:stringVariable"/>
                <xsd:element ref="masc-ex:externalOperationResult"/>
                <xsd:element ref="masc-ex:stringExpression"/>
                <xsd:element ref="masc-ex:stringExpressionRef"/>
            </xsd:choice>
            <xsd:attribute name="name" type="xsd:NCName" use="optional"/>
        </xsd:complexType>
        <!-- stringComparator -->
        <xsd:element name="stringComparator" type="masc-ex:stringComparatorType"/>
        <xsd:complexType name="stringComparatorType">
            <xsd:attribute name="type" type="masc-ex:stringComparatorTypeChoice" use="required"/>
        </xsd:complexType>
        <!-- timeExpression -->
        <xsd:element name="timeExpression" type="masc-ex:timeExpressionType"/>
        <xsd:complexType name="timeExpressionType">
            <xsd:choice>
                <xsd:element ref="masc-ex:timeConstant"/>
                <xsd:element ref="masc-ex:timeVariable"/>
```

```
                <xsd:element ref="masc-ex:externalOperationResult"/>
                <xsd:element ref="masc-ex:timeExpression"/>
                <xsd:element ref="masc-ex:timeExpressionRef"/>
                <xsd:sequence>
                        <xsd:element ref="masc-ex:timeOperator"/>
                        <xsd:element ref="masc-ex:timeExpression"/>
                </xsd:sequence>
        </xsd:choice>
        <xsd:attribute name="name" type="xsd:NCName" use="optional"/>
</xsd:complexType>
<!-- timeConstant -->
<xsd:element name="timeConstant" type="masc-ex:timeConstantType"/>
<xsd:complexType name="timeConstantType">
        <xsd:choice>
                <xsd:element ref="masc-ex:time"/>
                <xsd:element ref="masc-ex:date"/>
                <xsd:element ref="masc-ex:date_and_time"/>
                <xsd:element ref="masc-ex:time_duration"/>
                <xsd:element ref="masc-ex:gregorianYear"/>
                <xsd:element ref="masc-ex:gregorianYearMonth"/>
                <xsd:element ref="masc-ex:gregorianMonthDay"/>
                <xsd:element ref="masc-ex:gregorianMonth"/>
                <xsd:element ref="masc-ex:gregorianDay"/>
        </xsd:choice>
</xsd:complexType>
<!-- timeOperator -->
<xsd:element name="timeOperator" type="masc-ex:timeOperatorType"/>
<xsd:complexType name="timeOperatorType">
        <xsd:attribute name="type" type="masc-ex:timeOperatorTypeChoice" use="required"/>
</xsd:complexType>
<!-- timeComparator -->
<xsd:element name="timeComparator" type="masc-ex:timeComparatorType"/>
<xsd:complexType name="timeComparatorType">
        <xsd:attribute name="type" type="masc-ex:timeComparatorTypeChoice" use="required"/>
</xsd:complexType>
<!-- integerConstant -->
<xsd:element name="integerConstant" type="masc-ex:integerConstantType"/>
<xsd:complexType name="integerConstantType">
        <xsd:attribute name="value" type="xsd:int" use="required"/>
</xsd:complexType>
<!-- floatConstant -->
<xsd:element name="floatConstant" type="masc-ex:floatConstantType"/>
<xsd:complexType name="floatConstantType">
        <xsd:attribute name="value" type="xsd:float" use="required"/>
</xsd:complexType>
<!-- doubleConstant -->
<xsd:element name="doubleConstant" type="masc-ex:doubleConstantType"/>
<xsd:complexType name="doubleConstantType">
        <xsd:attribute name="value" type="xsd:double" use="required"/>
</xsd:complexType>
<!-- longConstant -->
<xsd:element name="longConstant" type="masc-ex:longConstantType"/>
<xsd:complexType name="longConstantType">
        <xsd:attribute name="value" type="xsd:long" use="required"/>
</xsd:complexType>
<!-- time -->
<xsd:element name="time" type="masc-ex:timeType"/>
<xsd:complexType name="timeType">
        <xsd:attribute name="value" type="xsd:time" use="required"/>
</xsd:complexType>
<!-- date -->
<xsd:element name="date" type="masc-ex:dateType"/>
<xsd:complexType name="dateType">
```

```
        <xsd:attribute name="value" type="xsd:date" use="required"/>
</xsd:complexType>
<!-- date_and_time -->
<xsd:element name="date_and_time" type="masc-ex:date_and_timeType"/>
<xsd:complexType name="date_and_timeType">
        <xsd:attribute name="value" type="xsd:dateTime" use="required"/>
</xsd:complexType>
<!-- time_duration -->
<xsd:element name="time_duration" type="masc-ex:time_durationType"/>
<xsd:complexType name="time_durationType">
        <xsd:attribute name="value" type="xsd:duration" use="required"/>
</xsd:complexType>
<!-- georgianYear -->
<xsd:element name="gregorianYear" type="masc-ex:gregorianYearType"/>
<xsd:complexType name="gregorianYearType">
        <xsd:attribute name="value" type="xsd:gYear" use="required"/>
</xsd:complexType>
<!-- georgianYearMonth -->
<xsd:element name="gregorianYearMonth" type="masc-ex:gregorianYearMonthType"/>
<xsd:complexType name="gregorianYearMonthType">
        <xsd:attribute name="value" type="xsd:gYearMonth" use="required"/>
</xsd:complexType>
<!-- georgianMonthDay -->
<xsd:element name="gregorianMonthDay" type="masc-ex:gregorianMonthDayType"/>
<xsd:complexType name="gregorianMonthDayType">
        <xsd:attribute name="value" type="xsd:gMonthDay" use="required"/>
</xsd:complexType>
<!-- georgianMonth -->
<xsd:element name="gregorianMonth" type="masc-ex:gregorianMonthType"/>
<xsd:complexType name="gregorianMonthType">
        <xsd:attribute name="value" type="xsd:gMonth" use="required"/>
</xsd:complexType>
<!-- georgianDay -->
<xsd:element name="gregorianDay" type="masc-ex:gregorianDayType"/>
<xsd:complexType name="gregorianDayType">
        <xsd:attribute name="value" type="xsd:gDay" use="required"/>
</xsd:complexType>
<!-- booleanVariable -->
<xsd:element name="booleanVariable" type="masc-ex:booleanVariableType"/>
<xsd:complexType name="booleanVariableType">
        <xsd:attribute name="bvName" type="xsd:QName" use="required"/>
</xsd:complexType>
<!-- arithmeticVariable -->
<xsd:element name="arithmeticVariable" type="masc-ex:arithmeticVariableType"/>
<xsd:complexType name="arithmeticVariableType">
        <xsd:attribute name="avName" type="xsd:QName" use="required"/>
</xsd:complexType>
<!-- arithmeticWithUnitVariable -->
<xsd:element name="arithmeticWithUnitVariable" type="masc-ex:arithmeticWithUnitVariableType"/>
<xsd:complexType name="arithmeticWithUnitVariableType">
        <xsd:attribute name="aWUVName" type="xsd:QName" use="required"/>
</xsd:complexType>
<!-- stringVariable -->
<xsd:element name="stringVariable" type="masc-ex:stringVariableType"/>
<xsd:complexType name="stringVariableType">
        <xsd:attribute name="svName" type="xsd:QName" use="required"/>
</xsd:complexType>
<!-- timeVariable -->
<xsd:element name="timeVariable" type="masc-ex:timeVariableType"/>
<xsd:complexType name="timeVariableType">
        <xsd:attribute name="tvName" type="xsd:QName" use="required"/>
</xsd:complexType>
<!-- arrayVariable -->
```

```
        <xsd:element name="arrayVariable" type="masc-ex:arrayVariableType"/>
        <xsd:complexType name="arrayVariableType">
                <xsd:attribute name="arrayVName" type="xsd:QName" use="required"/>
        </xsd:complexType>
        <!-- stringConstant -->
        <xsd:element name="stringConstant" type="masc-ex:stringConstantType"/>
        <xsd:complexType name="stringConstantType">
                <xsd:attribute name="value" type="xsd:string" use="required"/>
        </xsd:complexType>
        <!-- BooleanExpressionRef -->
        <xsd:element name="BooleanExpressionRef" type="masc-ex:BooleanExpressionRefType"/>
        <xsd:complexType name="BooleanExpressionRefType">
                <xsd:attribute name="name" type="xsd:QName" use="required"/>
        </xsd:complexType>
        <!-- arithmeticExpressionRef -->
        <xsd:element name="arithmeticExpressionRef" type="masc-ex:arithmeticExpressionRefType"/>
        <xsd:complexType name="arithmeticExpressionRefType">
                <xsd:attribute name="name" type="xsd:QName" use="required"/>
        </xsd:complexType>
        <!-- arithmeticWithUnitExpressionRef -->
        <xsd:element name="arithmeticWithUnitExpressionRef" type="masc-
ex:arithmeticWithUnitExpressionRefType"/>
        <xsd:complexType name="arithmeticWithUnitExpressionRefType">
                <xsd:attribute name="name" type="xsd:QName" use="required"/>
        </xsd:complexType>
        <!-- quantifiedExpressionRef -->
        <xsd:element name="quantifiedExpressionRef" type="masc-ex:quantifiedExpressionRefType"/>
        <xsd:complexType name="quantifiedExpressionRefType">
                <xsd:attribute name="name" type="xsd:QName" use="required"/>
        </xsd:complexType>
        <!-- monitoringCheckExpressionRef -->
        <xsd:element name="monitoringCheckExpressionRef" type="masc-ex:monitoringCheckExpressionRefType"/>
        <xsd:complexType name="monitoringCheckExpressionRefType">
                <xsd:attribute name="name" type="xsd:QName" use="required"/>
        </xsd:complexType>
        <!-- stringExpressionRef -->
        <xsd:element name="stringExpressionRef" type="masc-ex:stringExpressionRefType"/>
        <xsd:complexType name="stringExpressionRefType">
                <xsd:attribute name="name" type="xsd:QName" use="required"/>
        </xsd:complexType>
        <!-- timeExpressionRef -->
        <xsd:element name="timeExpressionRef" type="masc-ex:timeExpressionRefType"/>
        <xsd:complexType name="timeExpressionRefType">
                <xsd:attribute name="name" type="xsd:QName" use="required"/>
        </xsd:complexType>
        <!-- unsatisfiedConstraintRef -->
        <xsd:element name="unsatisfiedConstraintRef">
                <xsd:complexType>
                        <xsd:attribute name="name" type="xsd:QName" use="required"/>
                </xsd:complexType>
        </xsd:element>
        <!-- numberWithUnit -->
        <xsd:complexType name="numberWithUnit">
                <xsd:sequence>
                        <xsd:element ref="masc-ex:number"/>
                        <xsd:element ref="masc-ex:unit"/>
                </xsd:sequence>
        </xsd:complexType>
        <xsd:element name="unit" type="masc-ex:ontologicalType"/>
        <xsd:complexType name="ontologicalType">
                <xsd:attribute name="ontology" type="xsd:anyURI" use="required"/>
                <xsd:attribute name="type" type="xsd:string" use="required"/>
        </xsd:complexType>
```

```
        <xsd:element name="number" type="masc-ex:numberType"/>
        <xsd:complexType name="numberType">
                <xsd:attribute name="value" type="xsd:double" use="required"/>
        </xsd:complexType>
        <!-- numberWithUnitConstant -->
        <xsd:element name="numberWithUnitConstant" type="masc-ex:numberWithUnit"/>
        <!-- !!! numberWithUnitVariable is actually called arithmeticWithUnitVariable !!! -->
</xsd:schema>
```